



US006633963B1

(12) **United States Patent**
Ellison et al.

(10) **Patent No.:** US 6,633,963 B1
(45) **Date of Patent:** Oct. 14, 2003

(54) **CONTROLLING ACCESS TO MULTIPLE MEMORY ZONES IN AN ISOLATED EXECUTION ENVIRONMENT**

(75) Inventors: Carl M. Ellison, Portland, OR (US); Roger A. Golliver, Beaverton, OR (US); Howard C. Herbert, Phoenix, AZ (US); Derrick C. Lin, San Mateo, CA (US); Francis X. McKeen, Portland, OR (US); Gilbert Neiger, Portland, OR (US); Ken Reneris, Wilbraham, MA (US); James A. Sutton, Portland, OR (US); Shreekant S. Thakkar, Portland, OR (US); Millind Mittal, Palo Alto, CA (US)

(73) Assignee: Intel Corporation, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 462 days.

(21) Appl. No.: 09/618,489

(22) Filed: Jul. 18, 2000

Related U.S. Application Data

(60) Provisional application No. 60/198,225, filed on Mar. 31, 2000.

(51) Int. Cl.⁷ G06F 12/06; G06F 12/14

(52) U.S. Cl. 711/163; 711/153; 711/152; 711/170; 711/173

(58) Field of Search 711/1, 147, 151, 711/152, 153, 163, 164, 170, 173, 220; 709/213

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,037,214 A 7/1977 Birney et al.

4,278,837 A	7/1981	Best
4,521,852 A	6/1985	Guttag
5,022,077 A	6/1991	Bealkowski et al.
5,079,737 A *	1/1992	Hackbarth 711/164

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	1146715	10/2001
WO	WO 97/29567	8/1997
WO	WO9844402	10/1998
WO	WO9905600	2/1999
WO	WO9957863	11/1999
WO	WO0062232	10/2000
WO	WO 01/27723 A1	4/2001
WO	WO 01/27821 A2	4/2001
WO	WO 01/75565 A2	10/2001
WO	WO 01/75595 A2	10/2001

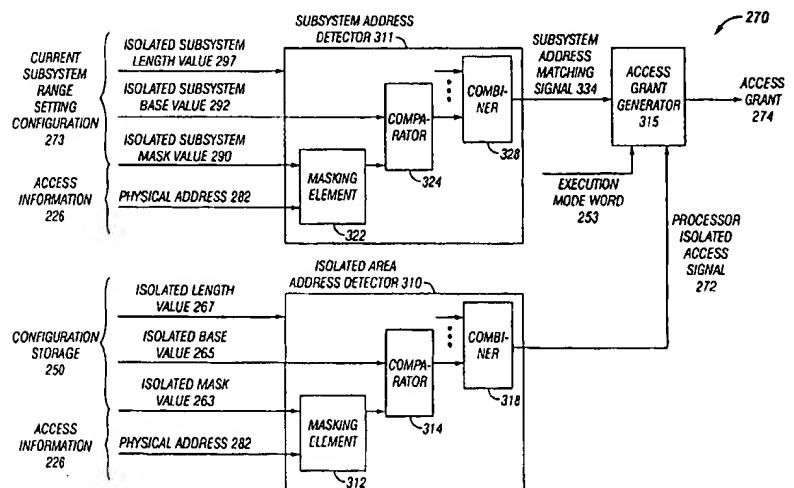
Primary Examiner—B. James Peikari

(74) Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

A processor having a normal execution mode and an isolated execution mode generates an access transaction. The access transaction is configured using a configuration storage that stores configuration settings. The configuration settings include a plurality of subsystem memory range settings defining memory zones. The access transaction also includes access information. A multi-memory zone access checking circuit, coupled to the configuration storage, checks the access transaction using at least one of the configuration settings and the access information. The multi-memory zone access checking circuit generates an access grant signal if the access transaction is valid.

44 Claims, 8 Drawing Sheets



U.S. PATENT DOCUMENTS

5,255,379 A	10/1993	Melo	6,014,745 A	1/2000	Ashe
5,293,424 A	3/1994	Holtey et al.	6,055,637 A	4/2000	Hudson et al.
5,386,552 A	1/1995	Garney	6,058,478 A	5/2000	Davis
5,421,006 A	5/1995	Jablon et al.	6,085,296 A	7/2000	Karkhanis et al.
5,459,869 A	10/1995	Spilo	6,125,430 A	9/2000	Noel et al.
5,473,692 A	12/1995	Davis	6,148,379 A	11/2000	Schimmel
5,479,509 A	12/1995	Ugon	6,158,546 A	12/2000	Hanson et al.
5,568,552 A	10/1996	Davis	6,175,925 B1	1/2001	Nardone et al.
5,615,263 A	3/1997	Takahashi	6,178,509 B1	1/2001	Nardone et al.
5,628,022 A	5/1997	Ueno et al.	6,192,455 B1	2/2001	Bogin et al.
5,657,445 A	8/1997	Pearce	6,205,550 B1	3/2001	Nardone et al.
5,717,903 A	2/1998	Bonola	6,249,872 B1 *	6/2001	Wildgrube et al. 713/200
5,729,760 A	3/1998	Poisner	6,272,533 B1 *	8/2001	Browne 709/213
5,737,760 A	* 4/1998	Grimmer et al. 711/163	6,282,651 B1	8/2001	Ashe
5,757,919 A	5/1998	Herbert et al.	6,282,657 B1	8/2001	Kaplan et al.
5,764,969 A	6/1998	Kahle et al.	6,292,874 B1 *	9/2001	Barnett 711/153
5,796,845 A	8/1998	Serikawa et al.	6,301,646 B1	10/2001	Hostetter
5,805,712 A	9/1998	Davis	6,321,314 B1	11/2001	Van Dyke
5,835,594 A	11/1998	Albrecht et al.	6,339,815 B1	1/2002	Feng et al.
5,844,986 A	12/1998	Davis	6,339,816 B1	1/2002	Bausch
5,852,717 A	12/1998	Bhide et al.	6,499,123 B1	12/2002	McFarland et al.
5,872,994 A	2/1999	Akiyama et al.	6,505,279 B1	1/2003	Phillips et al.
5,937,063 A	8/1999	Davis	2001/0021969 A1	9/2001	Burger et al.
5,978,481 A	11/1999	Ganesan et al.	2001/0027527 A1	10/2001	Khidekel et al.

* cited by examiner

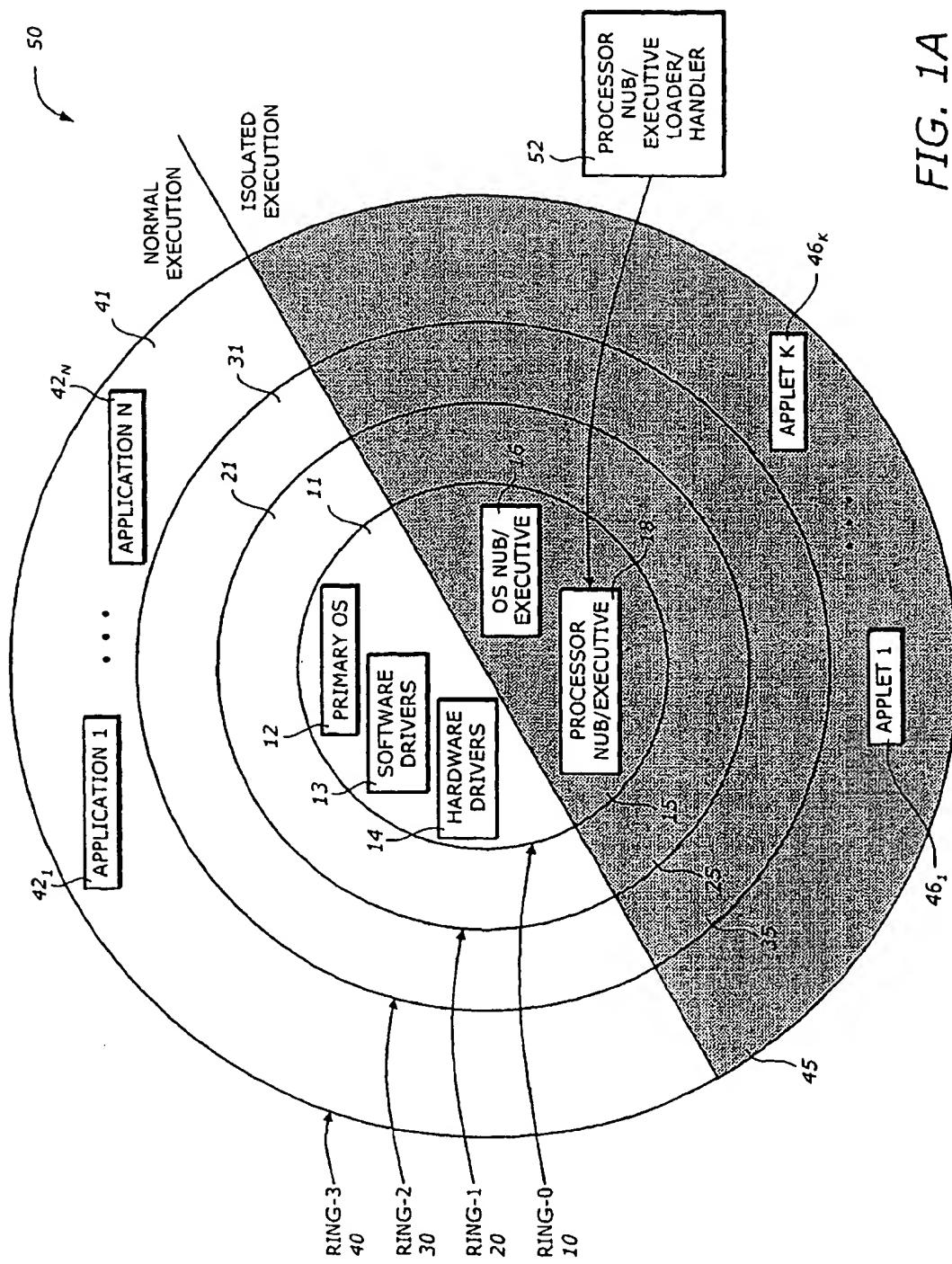


FIG. 1A

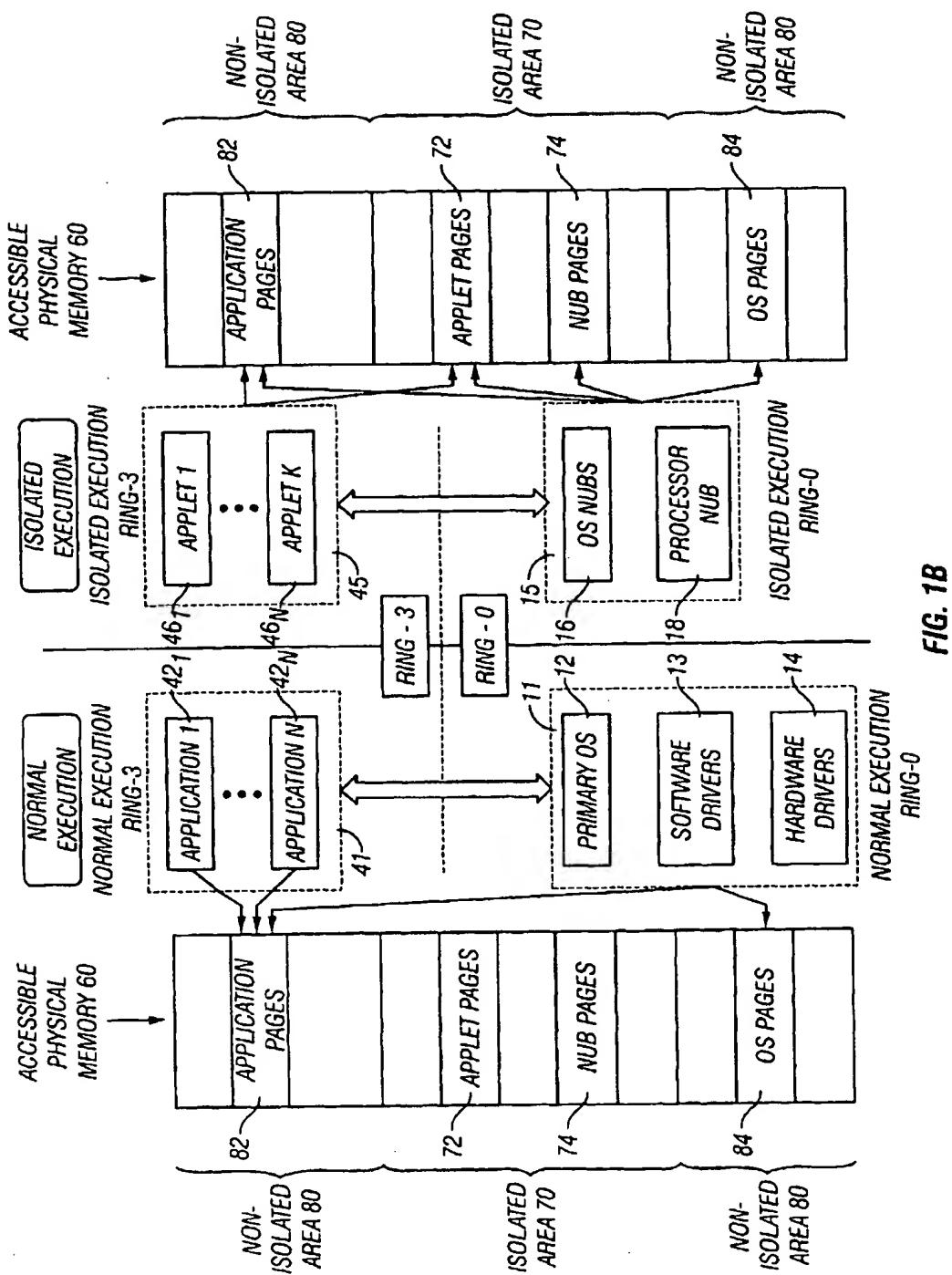


FIG. 1B

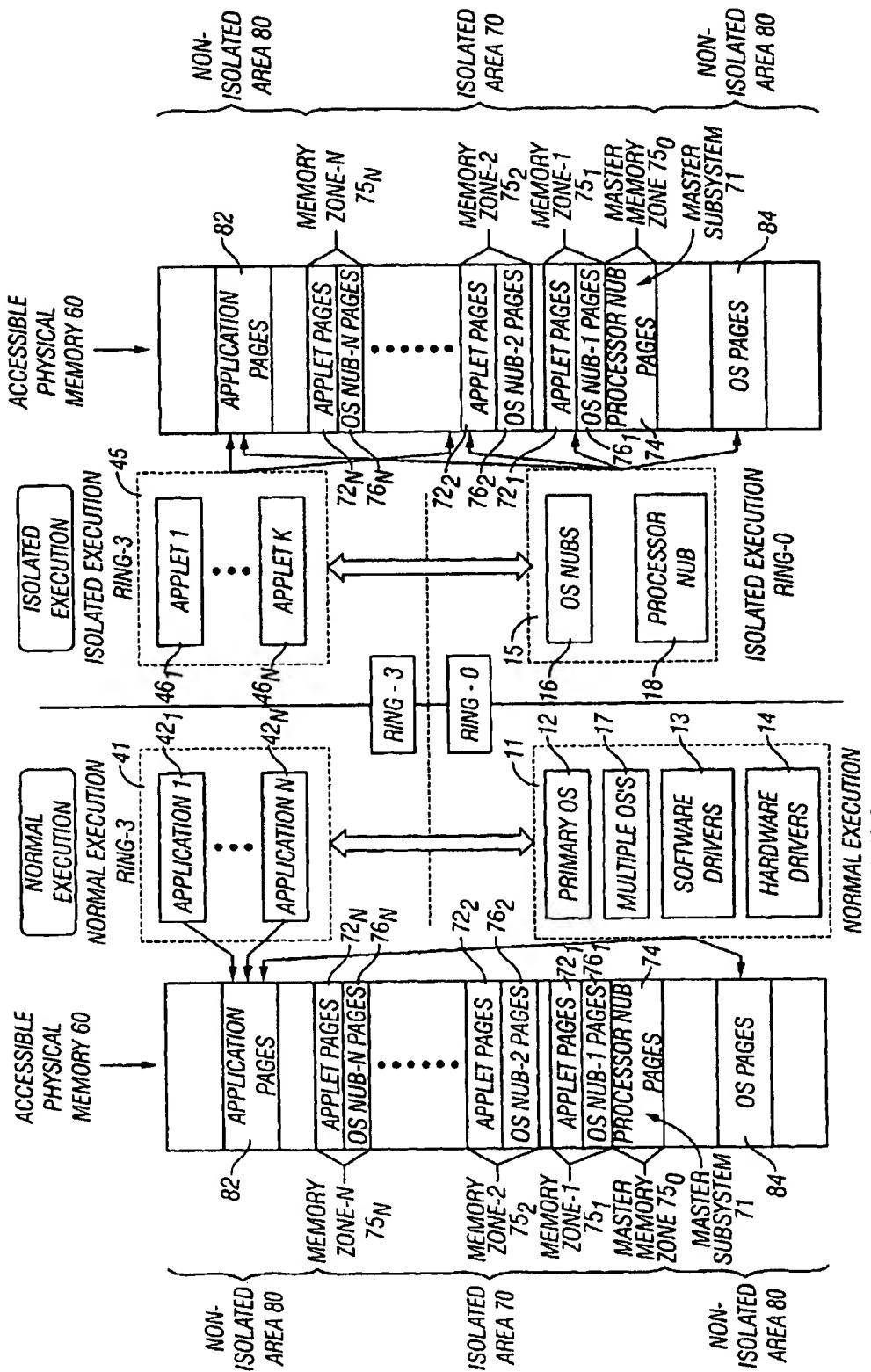


FIG. 1C

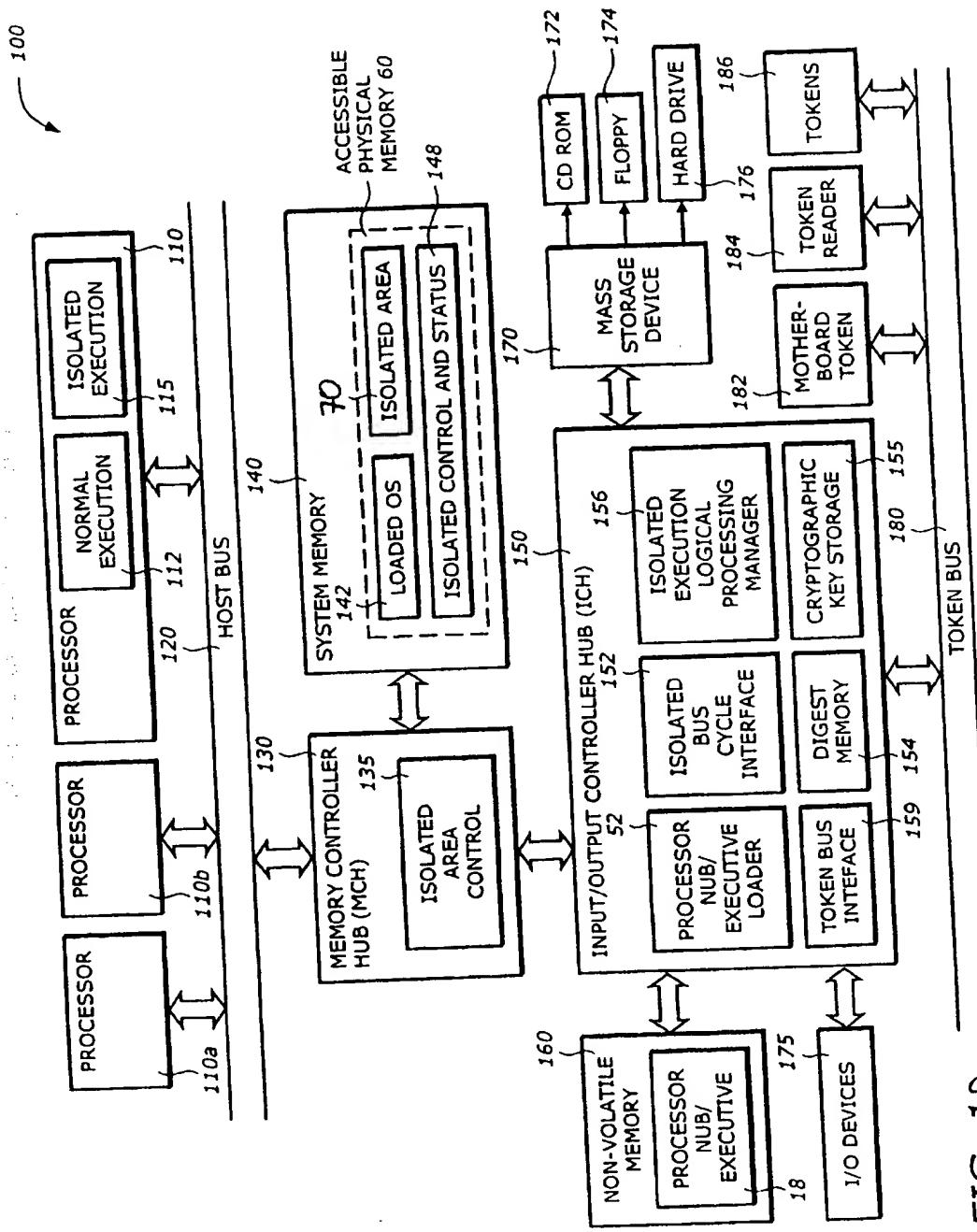


FIG. 1D

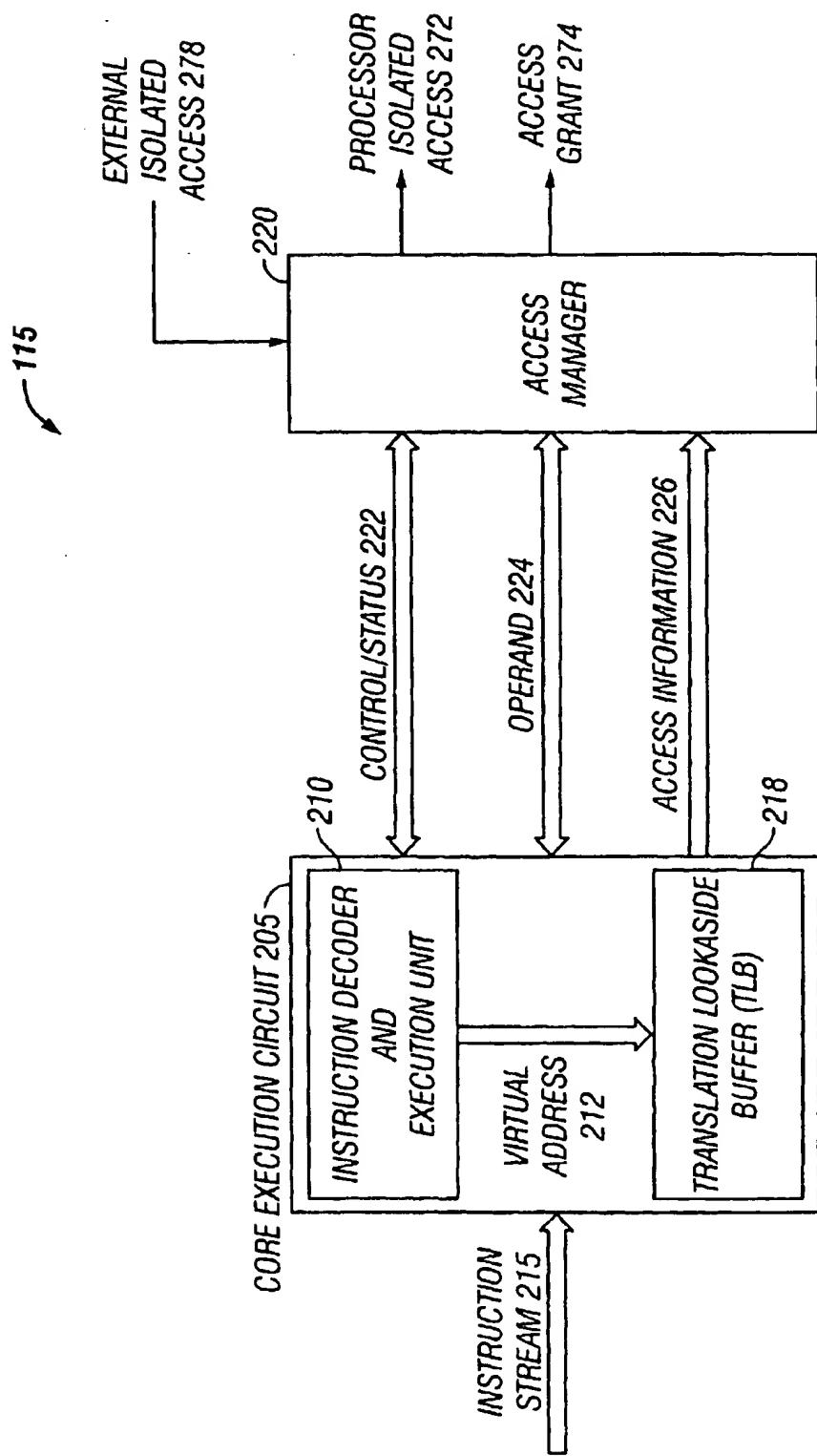
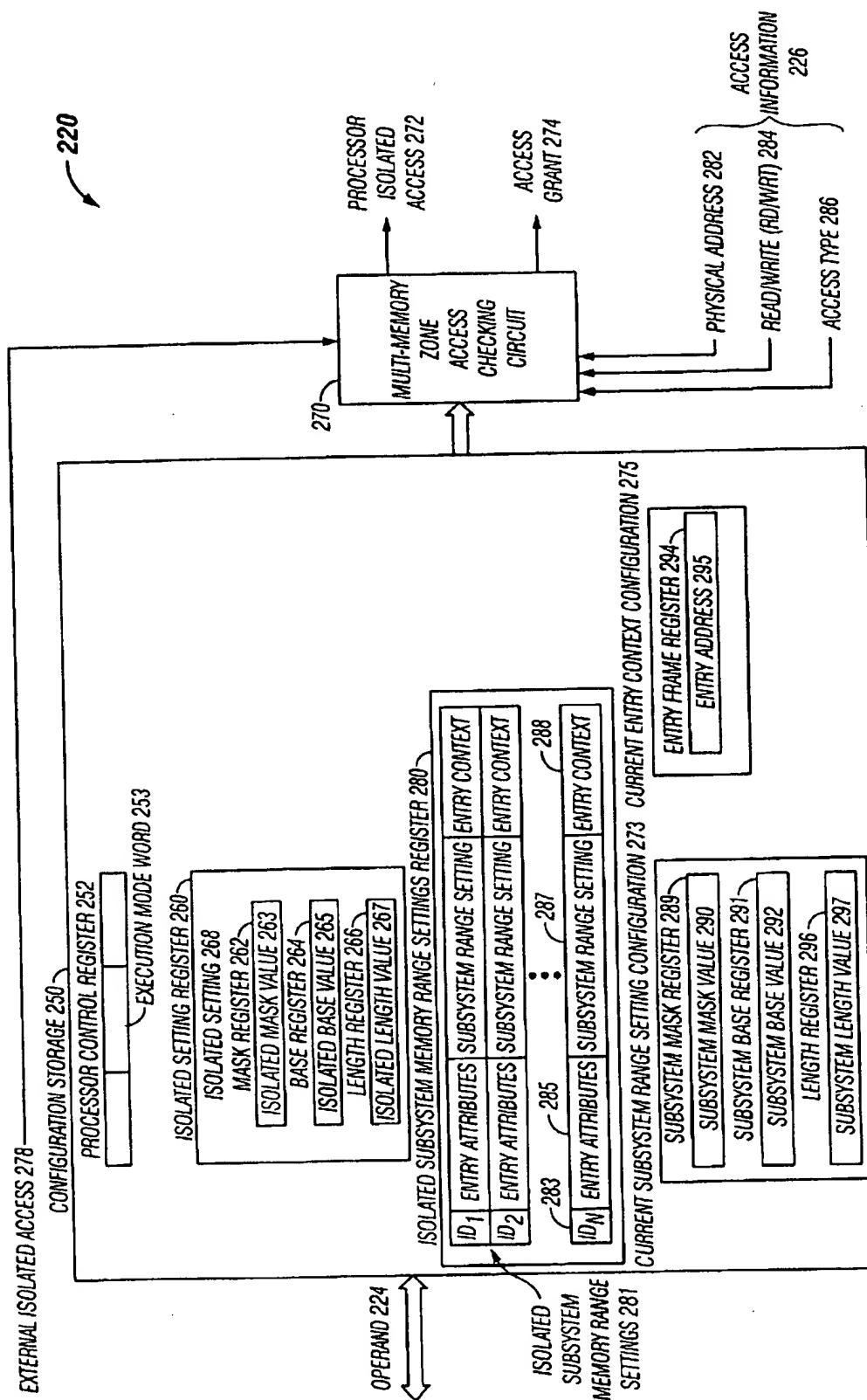


FIG. 2A



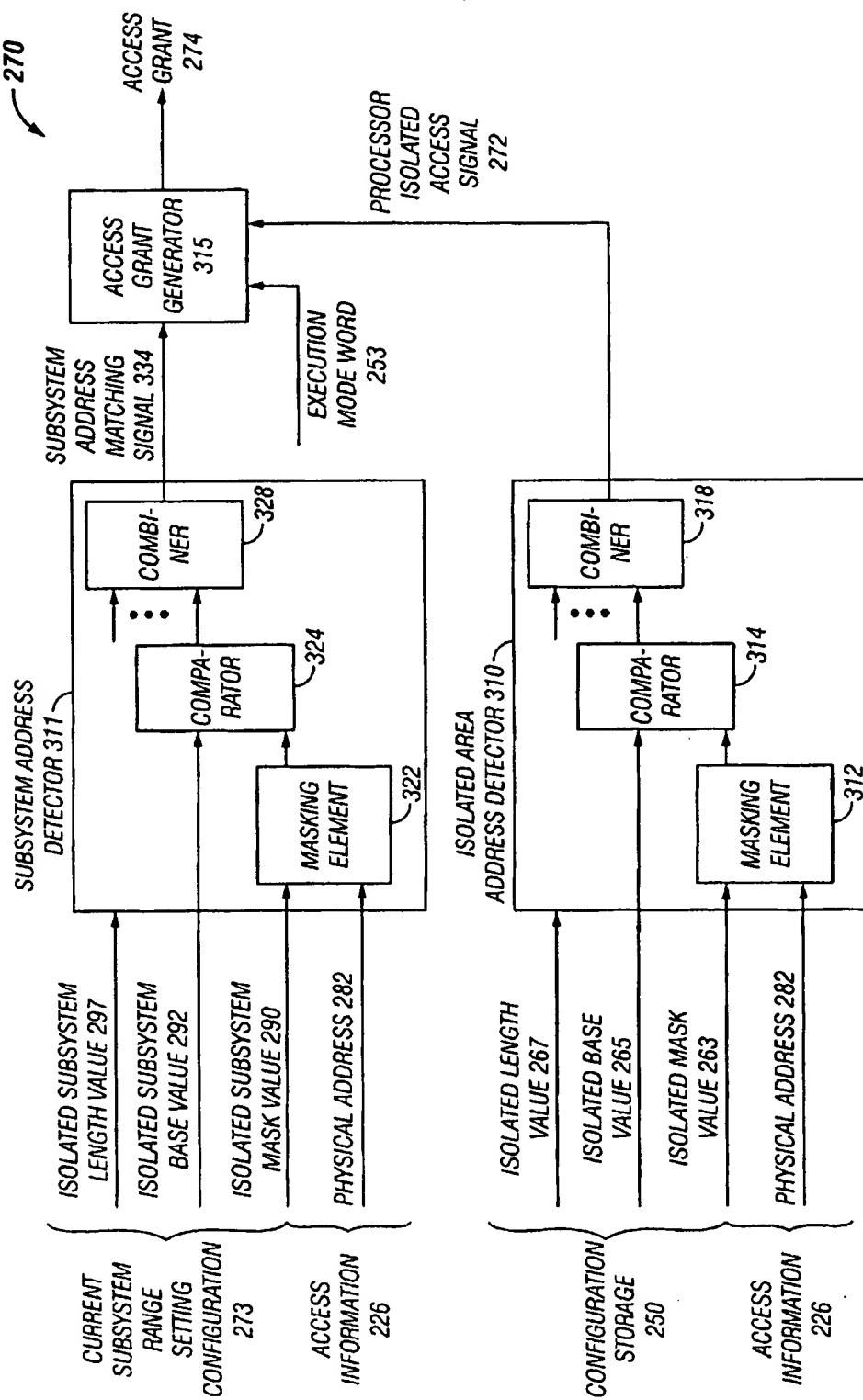


FIG. 3

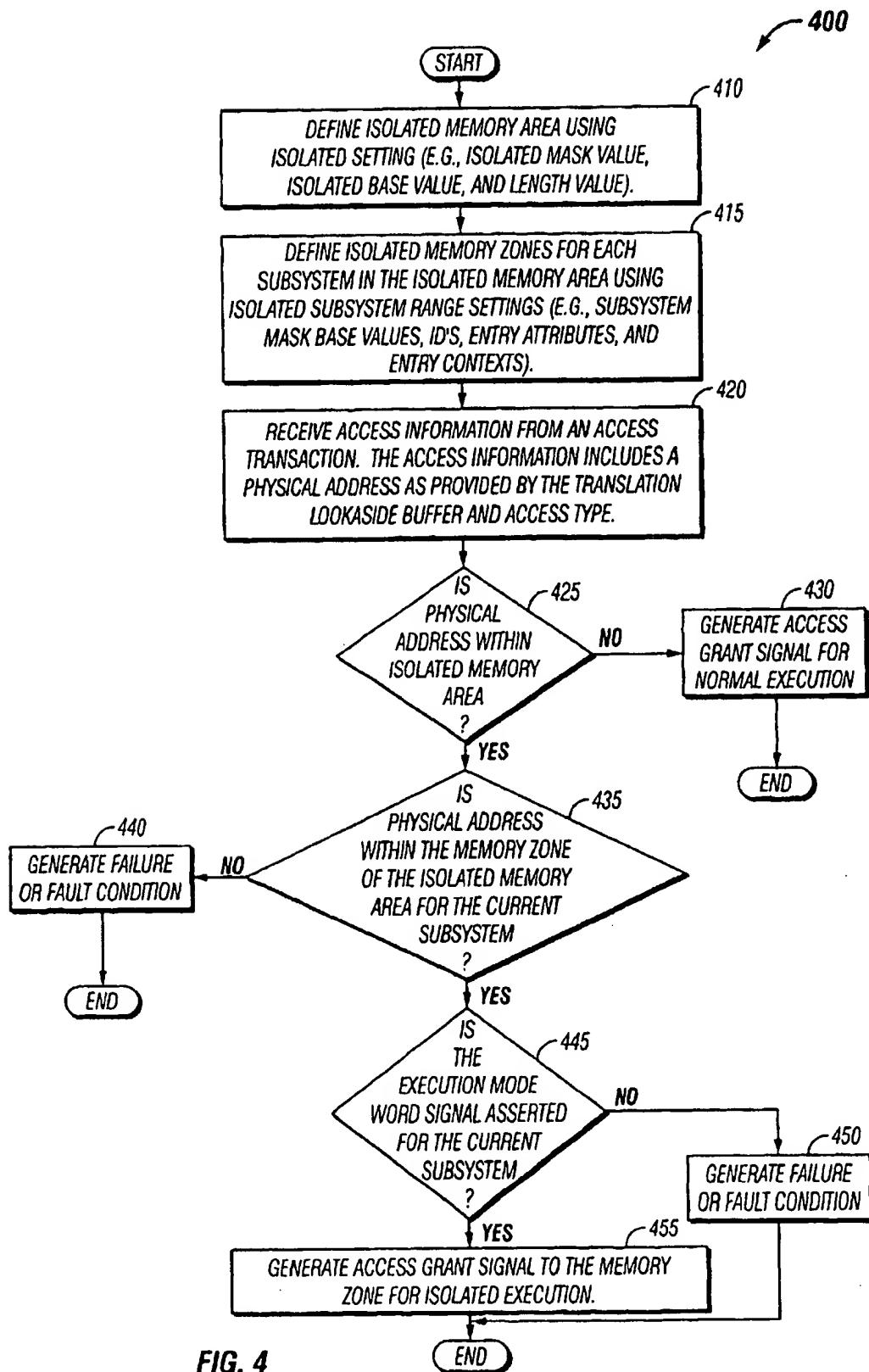


FIG. 4

**CONTROLLING ACCESS TO MULTIPLE
MEMORY ZONES IN AN ISOLATED
EXECUTION ENVIRONMENT**

RELATED APPLICATION

This application claims the benefit U.S. Provisional Patent Application No. 60/198,225 filed on Mar. 31, 2000.

BACKGROUND

1. Field of the Invention

This invention relates to microprocessors. In particular, the invention relates to processor security.

2. Description of Related Art

Advances in microprocessor and communication technologies have opened up many opportunities for applications that go beyond the traditional ways of doing business. Electronic commerce (E-commerce) and business-to-business (B2B) transactions are now becoming popular and are conducted throughout global markets at continuously increasing rates. Unfortunately, while modern microprocessor systems provide users convenient and efficient methods of doing business, communicating and transacting, they are also vulnerable for unscrupulous attacks. Examples of these attacks include virus, intrusion, security breach, and tampering, to name a few. Computer security, therefore, is becoming more and more important to protect the integrity of the computer systems and increase the trust of users.

Threats caused by unscrupulous attacks may be in a number of forms. An invasive remote-launched attack by hackers may disrupt the normal operation of a system connected to thousands or even millions of users. A virus program may corrupt code and/or data of a single-user platform.

Existing techniques to protect against attacks have a number of drawbacks. Anti-virus programs can only scan and detect known viruses. Security co-processors or smart cards using cryptographic or other security techniques have limitations in speed performance, memory capacity, and flexibility. Further, redesigning operating systems creates software compatibility issues and requires tremendous investment in development efforts.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

FIG. 1A is a diagram illustrating an operating system according to one embodiment of the invention.

FIG. 1B is a diagram illustrating the accessibility of various elements in the operating system and the processor according to one embodiment of the invention.

FIG. 1C is a diagram, similar to FIG. 1B, illustrating the accessibility of various elements in the operating system and the processor, in which, an isolated memory area includes a plurality of memory zones each of which is associated with a subsystem, respectively, according to one embodiment of the invention.

FIG. 1D is a diagram illustrating a computer system in which one embodiment of the invention can be practiced.

FIG. 2A is a diagram illustrating an embodiment of the isolated execution circuit shown in FIG. 1D according to one embodiment of the invention.

FIG. 2B is a diagram illustrating an access manager shown in FIG. 2A according to one embodiment of the invention.

FIG. 3 is a diagram illustrating a multi-memory zone access checking circuit according to one embodiment of the invention.

FIG. 4 is a flowchart illustrating a process to control access to multi-memory zones of the isolated memory area for isolated execution according to one embodiment of the invention.

DESCRIPTION

10 The present invention is a method, apparatus and system to control memory accesses to multiple memory zones in an isolated execution environment. A processor having a normal execution mode and an isolated execution mode generates an access transaction. The access transaction is configured using a configuration storage that stores configuration settings. The configuration settings include a plurality of subsystem memory range settings defining multiple memory zones. The access transaction also includes access information. A multi-memory zone access checking circuit, coupled to the configuration storage, checks the access transaction using at least one of the configuration settings and the access information. The multi-memory zone access checking circuit generates an access grant signal if the access transaction is valid.

20 In one embodiment, the configuration settings include an isolated setting, a plurality of subsystem memory range settings, and an execution mode word. The access information includes a physical address. The configuration storage includes a setting storage to store the isolated setting that defines an isolated memory area in a memory external to the processor. The isolated setting includes a base register, a mask register and a length register to store a base value, a mask value and a length value, respectively. A combination of at least two of the base, mask and length values can be used to form the isolated setting. The configuration storage further includes a subsystem memory range storage that stores the plurality of subsystem memory range settings. Each subsystem memory range setting defines a memory zone for a subsystem in the isolated memory area. Each subsystem memory range setting includes a subsystem mask value, a subsystem base value and a subsystem length value. A combination of at least two of the subsystem base, mask and length values can be used to define a memory zone. The subsystem memory range storage further includes ID values for each subsystem to identify each subsystem and the subsystem's associated memory zones as defined by the subsystem memory range setting. The configuration storage also includes a processor control register to store the execution mode word. The execution mode word is asserted as an execution mode word signal identifying a currently active subsystem and indicating that the processor is configured in the isolated execution mode. The isolated memory area is only accessible to the processor when it is configured in the isolated execution mode.

30 In one embodiment, the multi-memory zone access checking circuit includes a subsystem address detector. The subsystem address detector detects if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem. If the physical address is within the subsystem's associated memory zone, the subsystem address detector generates a subsystem address matching signal. An access grant generator is coupled to the subsystem address detector and the processor control register. The access grant generator generates an access grant signal if both the subsystem address matching signal and the execution mode

word signal, identifying the currently active subsystem operating in the isolated execution mode, are asserted. Thus, only if the physical address requested by a processor, operating in an isolated execution mode, is within the correct memory zone for the currently initialized subsystem will the access transaction be granted.

In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be appreciated to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well-known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention.

Architecture Overview

One principle for providing security in a computer system or platform is the concept of an isolated execution architecture. The isolated execution architecture includes logical and physical definitions of hardware and software components that interact directly or indirectly with an operating system of the computer system or platform. An operating system and the processor may have several levels of hierarchy, referred to as rings, corresponding to various operational modes. A ring is a logical division of hardware and software components that are designed to perform dedicated tasks within the operating system. The division is typically based on the degree or level of privilege, namely, the ability to make changes to the platform. For example, a ring-0 is the innermost ring, being at the highest level of the hierarchy. Ring-0 encompasses the most critical, privileged components. In addition, modules in Ring-0 can also access to lesser privileged data, but not vice versa. Ring-3 is the outermost ring, being at the lowest level of the hierarchy. Ring-3 typically encompasses users or applications level and has the least privilege protection. Ring-1 and ring-2 represent the intermediate rings with decreasing levels of privilege.

FIG. 1A is a diagram illustrating a logical operating architecture 50 according to one embodiment of the invention. The logical operating architecture 50 is an abstraction of the components of an operating system and the processor. The logical operating architecture 50 includes ring-0 10, ring-1 20, ring-2 30, ring-3 40, and a processor nub loader 52. The processor nub loader 52 is an instance of an processor executive (PE) handler. The PE handler is used to handle and/or manage a processor executive (PE) as will be discussed later. The logical operating architecture 50 has two modes of operation: normal execution mode and isolated execution mode. Each ring in the logical operating architecture 50 can operate in both modes. The processor nub loader 52 operates only in the isolated execution mode.

Ring-0 10 includes two portions: a normal execution Ring-0 11 and an isolated execution Ring-0 15. The normal execution Ring-0 11 includes software modules that are critical for the operating system, usually referred to as kernel. These software modules include primary operating system (e.g., kernel) 12, software drivers 13, and hardware drivers 14. The isolated execution Ring-0 15 includes an operating system (OS) nub 16 and a processor nub 18. The OS nub 16 and the processor nub 18 are instances of an OS executive (OSE) and processor executive (PE), respectively. The OSE and the PE are part of executive entities that operate in a protected environment associated with the isolated area 70 and the isolated execution mode. The processor nub loader 52 is a protected bootstrap loader code

held within a chipset in the system and is responsible for loading the processor nub 18 from the processor or chipset into an isolated area as will be explained later.

Similarly, ring-1 20, ring-2 30, and ring-3 40 include normal execution ring-1 21, ring-2 31, ring-3 41, and isolated execution ring-1 25, ring-2 35, and ring-3 45, respectively. In particular, normal execution ring-3 includes N applications 42₁ to 42_N, and isolated execution ring-3 includes K applets 46₁ to 46_K.

One concept of the isolated execution architecture is the creation of an isolated region in the system memory, referred to as an isolated area, which is protected by both the processor and chipset in the computer system. The isolated region may also be in cache memory, protected by a translation look aside (TLB) access check. Access to this isolated region is permitted only from a front side bus (FSB) of the processor, using special bus (e.g., memory read and write) cycles, referred to as isolated read and write cycles. The special bus cycles are also used for snooping. The isolated

read and write cycles are issued by the processor executing in an isolated execution mode. The isolated execution mode is initialized using a privileged instruction in the processor, combined with the processor nub loader 52. The processor nub loader 52 verifies and loads a ring-0 nub software module (e.g., processor nub 18) into the isolated area. The processor nub 18 provides hardware-related services for the isolated execution.

One task of the processor nub 18 is to verify and load the ring-0 OS nub 16 into the isolated area, and to generate the root of a key hierarchy unique to a combination of the platform, the processor nub 18, and the operating system nub 16. The processor nub 18 provides the initial set-up and low-level management of the isolated area including verification, loading, and logging of the operating system nub 16, and the management of a symmetric key used to protect the operating system nub's secrets. The processor nub 18 may also provide application programming interface (API) abstractions to low-level security services provided by other hardware.

The operating system nub 16 provides links to services in the primary OS 12 (e.g., the unprotected segments of the operating system), provides page management within the isolated area, and has the responsibility for loading ring-3 application modules 45, including applets 46₁ to 46_K, into protected pages allocated in the isolated area. The operating system nub 16 may also load ring-0 supporting modules.

The operating system nub 16 may choose to support paging of data between the isolated area and ordinary (e.g., non-isolated) memory. If so, then the operating system nub 16 is also responsible for encrypting and hashing the isolated area pages before evicting the page to the ordinary memory, and for checking the page contents upon restoration of the page. The isolated mode applets 46₁ to 46_K and their data are tamper-resistant and monitor-resistant from all software attacks from other applets, as well as from non-isolated-space applications (e.g., 42₁ to 42_N), dynamic link libraries (DLLs), drivers and even the primary operating system 12. Only the processor nub 18 or the operating system nub 16 can interfere with or monitor the applet's execution.

FIG. 1B is a diagram illustrating accessibility of various elements in the operating system 10 and the processor according to one embodiment of the invention. For illustration purposes, only elements of ring-0 10 and ring-3 40 are shown. The various elements in the logical operating architecture 50 access an accessible physical memory 60 according to their ring hierarchy and the execution mode.

The accessible physical memory 60 includes an isolated area 70 and a non-isolated area 80. The isolated area 70 includes applet pages 72 and nub pages 74. The non-isolated area 80 includes application pages 82 and operating system pages 84. The isolated area 70 is accessible only to elements of the operating system and processor operating in isolated execution mode. The non-isolated area 80 is accessible to all elements of the ring-0 operating system and processor.

The normal execution ring-0 11 including the primary OS 12, the software drivers 13, and the hardware drivers 14, can access both the OS pages 84 and the application pages 82. The normal execution ring-3, including applications 42₁ to 42_N, can access only to the application pages 82. Both the normal execution ring-0 11 and ring-3 41, however, cannot access the isolated area 70.

The isolated execution ring-0 15, including the OS nub 16 and the processor nub 18, can access both the isolated area 70, including the applet pages 72 and the nub pages 74, and the non-isolated area 80, including the application pages 82 and the OS pages 84. The isolated execution ring-3 45, including applets 46₁ to 46_K, can access only to the application pages 82 and the applet pages 72. The applets 46₁ to 46_K reside in the isolated area 70.

FIG. 1C is a diagram, similar to FIG. 1B, illustrating the accessibility of various elements in the operating system and the processor, in which, an isolated memory area 70 includes a plurality of memory zones (75₀-75_N) each of which is associated with a subsystem, respectively, according to one embodiment of the invention. For illustration purposes, only elements of ring-0 10 and ring-3 40 are shown. The various elements in the logical operating architecture 50 access an accessible physical memory 60 according to their ring hierarchy and the execution mode. The accessible physical memory 60 includes an isolated area 70 and a non-isolated area 80. In this embodiment shown in FIG. 1C, the isolated memory area 70 is divided up into a plurality of memory zones (75₀-75_N) each of which is associated with a subsystem. Particularly, there is a master memory zone 75₀ associated with a master subsystem 71 and a plurality of other memory zones (75₁-75_N) each of which is associated with an operating system to define a subsystem. This allows the present invention the ability to support concurrent multiple subsystems allowing for increased platform functionality.

To support the multiple subsystems, one of the subsystems is defined as the master subsystem 71. As illustrated in FIG. 1C, for exemplary purposes, the master subsystem 71 resides in master memory zone-0 75₀ and includes the processor nub 18 (processor executive (PE)) as processor nub pages 74. The processor nub 18 is loaded into one portion of the master memory zone-0 75₀. The master subsystem 71 is configured to include the entire isolated memory area 70 allowing it to control the memory used by the other zones. All other subsystems are configured as subsets of the master subsystem 70 and exclude the memory used by the processor nub 18. The master subsystem 71 is delegated with managing and creating the other subsystems and the other multiple memory zones (75₁-75_N). Particularly, the processor nub code executing in the master subsystem 71 provides the management and maintenance functions for the other subsystems such as allocating, configuring, initializing, and cleaning up the other memory zones. The master subsystem 71 is the only subsystem that can create additional subsystems by defining memory zones (e.g. 75₁-75_N) in the isolated memory area 70. Also, only code executing from the master subsystem may change the configuration of the other subsystems. Further, no subsystem

can examine or modify the contents of the master subsystem 71 or any other subsystem.

As shown in FIG. 1C, the present invention can have N number of memory zones to support N number of subsystems operating with N number of operating systems (OS's) and their corresponding OS nubs 16. Accordingly, the isolated memory area 70 can include multiple memory zones (75₁-75_N) each of which is associated with an operating system to define a subsystem. Only one memory zone and operating system is active at a time. Each OS nub corresponds to either the primary OS 12 or one of the multiple OS's 17, respectively. For example, a first subsystem could reside in memory zone-1 75₁ and includes the OS nub-1 pages 76₁ and applet pages 72₁. As an example, the OS nub-1 pages 76₁ could correspond to the OS nub for the primary OS 12 which could be, for example, a Windows™ operating system. Further, a second subsystem may reside in memory zone-2 75₂ and includes one of the multiple OS nubs 16 as OS nub-2 pages 76₂ and applet pages 72₂. As an example, a Linux™ operating system could be a second subsystem residing in memory zone-2 75₂. Furthermore, other operating systems can be located in the other remaining memory zones 75₃-75_N.

The non-isolated area 80 includes application pages 82 and operating system pages 84. The isolated area 70 is accessible only to elements of the operating system that is currently associated with the initialized subsystem and when the processor is operating in the isolated execution mode. The non-isolated area 80 is accessible to all elements of the ring-0 operating system and processor.

The normal execution ring-0 11 includes the primary OS 12, other multiple OS's 17, the software drivers 13, and the hardware drivers 14, all of which can access both the OS pages 84 and the application pages 82. The normal execution ring-3, including applications 42₁ to 42_N, can access only the application pages 82. Both the normal execution ring-0 11 and ring-3 41, however, cannot access the isolated area 70.

The isolated execution ring-0 15 includes the OS nubs 16 and the processor nub 18. Only the currently active subsystem associated with a particular memory zone (75₁-75_N) in the isolated memory area 70, as previously discussed, can access that memory zone and its associated OS Nub pages (76₁-76_N) and applet pages (72₁-72_N). Further, no subsystem can examine or modify the contents of the master subsystem 71 or any other subsystem. However, the isolated execution ring-0 15, including the OS nubs 16 and the processor nub 18, can access the non-isolated area 80, including the application pages 82 and the OS pages 84. Also, the isolated execution ring-3 45, including applets 46₁ to 46_K, can access the application pages 82.

FIG. 1D is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 includes a processor 110, a host bus 120, a memory controller hub (MCH) 130, a system memory 140, an input/output controller hub (ICH) 150, a non-volatile memory, or system flash, 160, a mass storage device 170, input/output devices 175, a token bus 180, a motherboard (MB) token 182, a reader 184, and a token 186. The MCH 130 may be integrated into a chipset that integrates multiple functionalities such as the isolated execution mode, host-to-peripheral bus interface, memory control. Similarly, the ICH 150 may also be integrated into a chipset together or separate from the MCH 130 to perform I/O functions. For clarity, not all the peripheral buses are shown. It is contemplated that the system 100 may also include

peripheral buses such as Peripheral Component Interconnect (PCI), accelerated graphics port (AGP), Industry Standard Architecture (ISA) bus, and Universal Serial Bus (USB), etc.

The processor 110 represents a central processing unit of any type of architecture, such as complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), or hybrid architecture. In one embodiment, the processor 110 is compatible with an Intel Architecture (IA) processor, such as the Pentium® processor series, the IA-32™ and the IA-64™. The processor 110 includes a normal execution mode 112 and an isolated execution circuit 115. The normal execution mode 112 is the mode in which the processor 110 operates in a non-protected environment, or a normal environment without the security features provided by the isolated execution mode. The isolated execution circuit 115 provides a mechanism to allow the processor 110 to operate in an isolated execution mode. The isolated execution circuit 115 provides hardware and software support for the isolated execution mode. This support includes configuration for isolated execution, definition of an isolated area, definition (e.g., decoding and execution) of isolated instructions, generation of isolated access bus cycles, and generation of isolated mode interrupts.

In one embodiment, the computer system 100 can be a single processor system, such as a desktop computer, which has only one main central processing unit, e.g. processor 110. In other embodiments, the computer system 100 can include multiple processors, e.g. processors 110, 110a, 110b, etc., as shown in FIG. 1D. Thus, the computer system 100 can be a multi-processor computer system having any number of processors. For example, the multi-processor computer system 100 can operate as part of a server or workstation environment. The basic description and operation of processor 110 will be discussed in detail below. It will be appreciated by those skilled in the art that the basic description and operation of processor 110 applies to the other processors 110a and 110b, shown in FIG. 1D, as well as any number of other processors that may be utilized in the multi-processor computer system 100 according to one embodiment of the present invention.

The processor 110 may also have multiple logical processors. A logical processor, sometimes referred to as a thread, is a functional unit within a physical processor having an architectural state and physical resources allocated according to some partitioning policy. Within the context of the present invention, the terms "thread" and "logical processor" are used to mean the same thing. A multi-threaded processor is a processor having multiple threads or multiple logical processors. A multi-processor system (e.g., the system comprising the processors 110, 110a, and 110b) may have multiple multi-threaded processors.

The host bus 120 provides interface signals to allow the processor 110 or processors 110, 110a, and 110b to communicate with other processors or devices, e.g., the MCH 130. In addition to normal mode, the host bus 120 provides an isolated access bus mode with corresponding interface signals for memory read and write cycles when the processor 110 is configured in the isolated execution mode. The isolated access bus mode is asserted on memory accesses initiated while the processor 110 is in the isolated execution mode. The isolated access bus mode is also asserted on instruction pre-fetch and cache write-back cycles if the address is within the isolated area address range and the processor 110 is initialized in the isolated execution mode.

The processor 110 responds to snoop cycles to a cached address within the isolated area address range if the isolated access bus cycle is asserted and the processor 110 is initialized into the isolated execution mode.

The MCH 130 provides control and configuration of memory and input/output devices such as the system memory 140 and the ICH 150. The MCH 130 provides interface circuits to recognize and service isolated access assertions on memory reference bus cycles, including isolated memory read and write cycles. In addition, the MCH 130 has memory range registers (e.g., base and length registers) to represent the isolated area in the system memory 140. Once configured, the MCH 130 aborts any access to the isolated area that does not have the isolated access bus mode asserted.

The system memory 140 stores system code and data. The system memory 140 is typically implemented with dynamic random access memory (DRAM) or static random access memory (SRAM). The system memory 140 includes the accessible physical memory 60 (shown in FIG. 1B). The accessible physical memory includes a loaded operating system 142, the isolated area 70 (shown in FIG. 1B), and an isolated control and status space 148. The loaded operating system 142 is the portion of the operating system that is loaded into the system memory 140. The loaded OS 142 is typically loaded from a mass storage device via some boot code in a boot storage such as a boot read only memory (ROM). The isolated area 70, as shown in FIG. 1B, is the memory area that is defined by the processor 110 when operating in the isolated execution mode. Access to the isolated area 70 is restricted and is enforced by the processor 110 and/or the MCH 130 or other chipset that integrates the isolated area functionalities. The isolated control and status space 148 is an input/output (I/O)-like, independent address space defined by the processor 110 and/or the MCH 130. The isolated control and status space 148 contains mainly the isolated execution control and status registers. The isolated control and status space 148 does not overlap any existing address space and is accessed using the isolated bus cycles. The system memory 140 may also include other programs or data which are not shown.

The ICH 150 represents a known single point in the system having the isolated execution functionality. For clarity, only one ICH 150 is shown. The system 100 may have many ICH's similar to the ICH 150. When there are multiple ICH's, a designated ICH is selected to control the isolated area configuration and status. In one embodiment, this selection is performed by an external strapping pin. As is known by one skilled in the art, other methods of selecting can be used, including using programmable configuring registers. The ICH 150 has a number of functionalities that are designed to support the isolated execution mode in addition to the traditional I/O functions. In particular, the ICH 150 includes an isolated bus cycle interface 152, the processor nub loader 52 (shown in FIG. 1A), a digest memory 154, a cryptographic key storage 155, an isolated execution logical processing manager 156, and a token bus interface 159.

The isolated bus cycle interface 152 includes circuitry to interface to the isolated bus cycle signals to recognize and service isolated bus cycles, such as the isolated read and write bus cycles. The processor nub loader 52, as shown in FIG. 1A, includes a processor nub loader code and its digest (e.g., hash) value. The processor nub loader 52 is invoked by execution of an appropriate isolated instruction (e.g., Iso-Init) and is transferred to the isolated area 70. From the isolated area 80, the processor nub loader 52 copies the

processor nub 18 from the system flash (e.g., the processor nub code 18 in non-volatile memory 160) into the isolated area 70, verifies and logs its integrity, and manages a symmetric key used to protect the processor nub's secrets. In one embodiment, the processor nub loader 52 is implemented in read only memory (ROM). For security purposes, the processor nub loader 52 is unchanging, tamper-resistant and non-substitutable. The digest memory 154, typically implemented in RAM, stores the digest (e.g., hash) values of the loaded processor nub 18, the operating system nub 16, and any other critical modules (e.g., ring-0 modules) loaded into the isolated execution space. The cryptographic key storage 155 holds a symmetric encryption/decryption key that is unique for the platform of the system 100. In one embodiment, the cryptographic key storage 155 includes internal fuses that are programmed at manufacturing. Alternatively, the cryptographic key storage 155 may also be created with a random number generator and a strap of a pin. The isolated execution logical processing manager 156 manages the operation of logical processors operating in isolated execution mode. In one embodiment, the isolated execution logical processing manager 156 includes a logical processor count register that tracks the number of logical processors participating in the isolated execution mode. The token bus interface 159 interfaces to the token bus 180. A combination of the processor nub loader digest, the processor nub digest, the operating system nub digest, and optionally additional digests, represents the overall isolated execution digest, referred to as an isolated digest. The isolated digest is a fingerprint identifying the ring-0 code controlling the isolated execution configuration and operation. The isolated digest is used to attest or prove the state of the current isolated execution.

The non-volatile memory 160 stores non-volatile information. Typically, the non-volatile memory 160 is implemented in flash memory. The non-volatile memory 160 includes the processor nub 18.

The processor nub 18 provides the initial set-up and low-level management of the isolated area 70 (in the system memory 140), including verification, loading, and logging of the operating system nub 16, and the management of the symmetric key used to protect the operating system nub's secrets. The processor nub 18 may also provide application programming interface (API) abstractions to low-level security services provided by other hardware. The processor nub 18 may also be distributed by the original equipment manufacturer (OEM) or operating system vendor (OSV) via a boot disk.

The mass storage device 170 stores archive information such as code (e.g., processor nub 18), programs, files, data, applications (e.g., applications 42₁ to 42_N), applets (e.g., applets 46₁ to 46_K) and operating systems. The mass storage device 170 may include compact disk (CD) ROM 172, floppy diskettes 174, and hard disk 176, and any other magnetic or optic storage devices. The mass storage device 170 provides a mechanism to read machine-readable media.

I/O devices 175 may include any I/O devices to perform I/O functions. Examples of I/O devices 175 include controller for input devices (e.g., keyboard, mouse, trackball, pointing device), media card (e.g., audio, video, graphics), network card, and any other peripheral controllers.

The token bus 180 provides an interface between the ICH 150 and various tokens in the system. A token is a device that performs dedicated input/output functions with security functionalities. A token has characteristics similar to a smart card, including at least one reserved-purpose public/private

key pair and the ability to sign data with the private key. Examples of tokens connected to the token bus 180 include a motherboard token 182, a token reader 184, and other portable tokens 186 (e.g., smart card). The token bus interface 159 in the ICH 150 connects through the token bus 180 to the ICH 150 and ensures that when commanded to prove the state of the isolated execution, the corresponding token (e.g., the motherboard token 182, the token 186) signs only valid isolated digest information. For purposes of security, the token should be connected to the digest memory.

When implemented in software, the elements of the present invention are the code segments to perform the necessary tasks. The program or code segments can be stored in a machine readable medium, such as a processor readable medium, or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable medium" may include any medium that can store or transfer information. Examples of the processor readable medium include an electronic circuit, a semiconductor memory device, a ROM, a flash memory, an erasable programmable ROM (EPROM), a floppy diskette, a compact disk CD-ROM, an optical disk, a hard disk, a fiber optic medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, Intranet, etc.

Controlling Access to Multiple Memory Zones in an Isolated Execution Environment

FIG. 2A is a diagram illustrating an embodiment of the isolated execution circuit 115 shown in FIG. 1D according to one embodiment of the invention. The isolated execution circuit 115 includes a core execution circuit 205 and an access manager 220.

The core execution circuit 205 includes an instruction decoder and execution unit 210 and a translation lookaside buffer (TLB) 218. The instruction decoder and execution unit 210 receives an instruction stream 215 from an instruction fetch unit. The instruction stream 215 includes a number of instructions. The instruction decoder and execution unit 210 decodes the instructions and executes the decoded instructions. These instructions may be at the micro- or macro- level. The instruction decoder and execution unit 210 may be a physical circuit or an abstraction of a process of decoding and execution of instructions. In addition, the instructions may include isolated instructions and non-isolated instructions. The instruction decoder and execution unit 210 generates a virtual address 212 when there is an access transaction, caused by the execution of instructions. The TLB 218 translates the virtual address 212 into a physical address.

The core execution circuit 205 interfaces with the access manager 220 via control/status information 222, operand 224, and access information 226. The control/status information 222 includes control bits to manipulate various elements in the access manager 220 and status data from the access manager 220. The operand 224 includes data to be written to and read from the access manager 220. The access information 226 includes address (e.g., the physical address provided by the TLB 218), read/write, and access type information.

The access manager 220 receives and provides the control/status information 222 and the operand 224, and

receives the access information 226 from the core execution circuit 205 as a result of instruction execution, and further receives an external isolated access signal 278 from another processor in the system. The external isolated access signal 278 is asserted when another processor in the system attempts to access the isolated memory area. The access manager 220 generates a processor isolated access signal 272 and an access grant signal 274. The processor isolated access signal 272 may be used to generate an isolated bus cycle sent to devices (e.g., chipsets) external to the processor 110 to indicate that the processor 110 is executing an isolated mode instruction. The processor isolated access signal 272 and the access grant signal 274 may also be used internally by the processor 110 to control and monitor other isolated or non-isolated activities.

FIG. 2B is a diagram illustrating the access manager 220 shown in FIG. 2A according to one embodiment of the invention. The access manager 220 includes a configuration storage 250 and a multi-memory zone access checking circuit 270. The access manager 220 exchanges the operand 224 with and receives the access information 226 from the core execution circuit 205 shown in FIG. 2A. The access manager 220 also receives the external isolated access signal 278 from another processor as previously shown in FIG. 2A. The access information 226 includes a physical address 282, a read/write (RD/WR#) signal 284 and an access type 286. The access information 226 is generated during an access transaction by the processor 110. The access type 286 indicates a type of access, including a memory reference, an input/output (I/O) reference, and a logical processor access. The logical processor access includes a logical processor entry to an isolated enabled state, and a logical processor withdrawal from an isolated enabled state.

The configuration storage 250 stores configuration parameters to configure an access transaction generated by the processor 110. The processor 110 has a normal execution mode and an isolated execution mode. The access transaction includes access information 226 as discussed above. The configuration storage 250 receives the operand 224 from the instruction decoder and execution unit 210 (FIG. 2A) and includes a processor control register 252, an isolated setting register 260, and an isolated subsystem memory range setting register 280. The processor control register 252 contains an execution mode word 253. The execution mode word 253 is asserted as an execution mode word signal when the processor 110 is configured in the isolated execution mode. In one embodiment, the execution mode word 253 stores the ID of the currently active subsystem operating in the isolated execution mode or an indicator that the processor is not currently in isolated execution mode.

The isolated setting register 260 contains an isolated setting 268. The isolated setting 268 defines the isolated memory area (e.g. the isolated area 70 in the accessible physical memory 60 shown in FIGS. 1B and 1C). The isolated setting register 260 includes a mask register 262, a base register 264, and a length register 266. The mask register 262 stores an isolated mask value 263. The base register 264 stores an isolated base value 265. The length register 266 stores an isolated length value 267. The isolated mask, base, and length values 263, 265, and 267 form the isolated setting 268 and are used to define the isolated memory area. The isolated memory area may be defined by using any combination of the mask, base, and length values 263, 265 and 267. For example, the base value 265 corresponds to the starting address of the isolated memory area, while the sum of the base value 265 and the length value 267 corresponds to the ending address of the isolated memory area.

The isolated subsystem memory range setting register 280 stores the isolated subsystem memory range settings 281. The isolated subsystem memory range settings 281 can be represented as a table having a plurality of rows and columns that store the settings for the subsystems within the isolated area. This table may represent the settings for all the possible subsystems in the isolated memory area, or it may represent a cache of settings for selected subsystems. Each row corresponds to a different subsystem and has a number of column entries that describe the subsystem and the subsystem's associated memory zone (75_0-75_N) in the isolated memory area 70 (FIG. 1C). Each row includes an ID value 283, an entry attribute 285, a subsystem range setting 287, and an entry context 288. The ID value is a unique identifier for each subsystem. The entry attribute 285 includes a plurality of values including page size (such as large or normal sized), read or write access, and a master subsystem flag.

Each subsystem identified by the isolated subsystem memory range settings 281 includes a subsystem range setting 287 having a subsystem mask value, a subsystem base value, and a subsystem length value, as will be discussed in more detail, that defines a memory zone (75_0-75_N) for a respective subsystem in the isolated memory area 70 (FIG. 1C). Each subsystem identified by the isolated subsystem memory range settings 281 also includes an entry context 288. The entry context 288 is used for entry by a subsystem into its associated memory zone for isolated execution.

One subsystem is defined as the master subsystem, and given a pre-defined fixed ID 283. This master subsystem is configured to include the entire isolated memory area. The processor nub is loaded into one portion of the master subsystem memory. All other subsystems are configured as subsets of the master subsystem, and exclude the memory used by the processor nub. The code executing in the master subsystem provides the management and maintenance functions for the other subsystems. Only code executing from the master subsystem may change the content of the isolated subsystem memory range setting register 280, thereby changing the configuration of the other subsystems.

When a master subsystem is initialized it fills in a row of the isolated subsystem memory range setting register 280 for itself including: setting its predetermined ID value 283, setting the master subsystem flag as a fixed entry in the entry attributes 285 to identify itself as the master subsystem, defining its associated memory zone using the subsystem range setting 287, and setting its entry address in the entry context 288. The isolated subsystem memory range setting register 280 is controlled by the master subsystem executive which corresponds to the processor nub 18 (or processor executive (PE)) of the master subsystem 71 (FIG. 1C). As the master subsystem initializes new subsystems it creates new rows in the isolated subsystem memory range setting register 280 defining each new subsystem.

To pass control from a first subsystem to a second subsystem, a subsystem control change instruction is issued. The subsystem control change instruction uses the second subsystem ID which is called as a parameter. The processor checks the IDs 283 of the isolated subsystem memory range settings 281 for a match, and if a match is found, the processor extracts the subsystem range setting 287 (e.g. subsystem mask value, subsystem base value, and subsystem length value to define the memory zone of the second subsystem) and the entry context field 288 of the second subsystem and copies them into the current subsystem range setting configuration register 273 and the current entry

context configuration register 275, respectively. These new configuration registers represent the configuration for the currently active subsystem, and are used to drive the multi-memory zone access checking circuit 270, as will be discussed.

The current subsystem range setting configuration register 273 includes a subsystem mask register 289, a subsystem base register 291, and a subsystem length register 296. The subsystem mask register 289 stores a subsystem mask value 290. The subsystem base register 291 stores a subsystem base value 292. The subsystem length register 296 stores a subsystem length value 297. The memory zone (75_0-75_N) of the isolated memory area 70 (FIG. 1C) for the currently active subsystem may be defined by any combination of the subsystem mask, base, and length values 290, 292, and 297. For example, the base value 292 corresponds to the starting address of the memory zone for the currently active subsystem, while the sum of the base value 292 and the length value 297 corresponds to the ending address of the memory zone for the currently active subsystem. The current entry context configuration register 275 includes an entry frame register 294 to store an entry frame address 295 copied from the entry context field 288. The entry address 295 is the physical address of the entry frame.

The processor can then enter into an isolated execution mode (e.g. iso_enter) under the control of the second subsystem. To do this, the processor loads an entry frame into a frame register set. The entry frame is pointed to by the entry address 295 in the entry frame register 296. The processor is now configured in an isolated execution mode under the control of the second subsystem, which is now the currently active subsystem, the memory zone of which is identified by the values stored in the current subsystem range setting configuration register 273, as previously discussed.

However, if an ID match does not occur, and a corresponding row of the isolated subsystem memory range settings 281 does not exist for the second subsystem, the processor checks the isolated subsystem memory range settings 281 for the master subsystem, which is identified by the set master subsystem flag in the entry attributes 285. The processor loads the parameters of the corresponding row of the subsystem isolated subsystem memory range settings 281 for the master subsystem and passes control to the master subsystem. Upon entry of the master subsystem, the processor determines that the second subsystem parameter ID is not for the master subsystem, but a different subsystem. If the second subsystem exists, the master subsystem defines the second subsystem in the isolated subsystem memory range settings 281 by creating a new row for the second subsystem. If the memory range setting register 280 is a cache, with fewer entries than the possible subsystems, and there are no unused entries available, then the master subsystem may evict one of the existing entries to replace it with the new row for the second subsystem. Only the master subsystem entry cannot be replaced. Control is then passed from the master subsystem (i.e. the first subsystem) to the second subsystem as previously discussed. However, if the second subsystem does not exist, the processor generates a failure or fault condition.

The multi-memory zone access checking circuit 270 checks the access transaction using at least one of the configuration parameters (e.g. the execution mode word 253 and the current subsystem range setting configuration 273) and the access information 226. The multi-memory zone access checking circuit 270 generates the processor isolated access signal 272 and the access grant signal 274. The

processor isolated access signal 272 is asserted when the processor 110 attempts a memory access to a physical address within the isolated memory area 70. The access grant signal 274 indicates whether a memory access request can be granted. The access grant signal 274 is asserted if the requested memory address is outside the isolated memory range, OR, the processor is currently executing in isolated mode AND the address is within the currently active isolated subsystem memory range.

FIG. 3 is a diagram illustrating the multi-memory zone access checking circuit 270 to manage accesses for isolated execution according to one embodiment of the invention. The multi-memory zone access checking circuit 270 includes an isolated area address detector 310, a subsystem address detector 311 and an access grant generator 315.

The isolated area address detector 310 receives the isolated setting 268 (e.g., the isolated mask value 263, the isolated base value 265, the isolated length value 267) from the configuration storage 250 in FIG. 2B. The isolated area address detector 310 detects if the physical address 282 is within the isolated memory area defined by the isolated setting 268. The isolated area address detector 310 includes a masking element 312, a comparator 314, and a combiner 318. The masking element 312 masks the physical address 282 with the isolated mask value 263. In one embodiment, the masking element 312 performs a logical AND operation. The comparator 314 compares the result of the masking operation done by the masking element 312 and the isolated base value 265, and generates a comparison result. In other embodiments, the comparator 314 can also use the isolated length value 267 in generating the comparison result. The combiner 318 combines the comparison result with other conditions to generate the processor isolated access signal 272. The processor isolated access signal 272 is asserted when the physical address 282 is within the isolated memory area as defined by the isolated mask and base values 263 and 265, and in some embodiments, the length value 267.

The subsystem address detector 311 receives the values of the current subsystem range setting configuration register 273 (e.g., the isolated subsystem mask value 290, the isolated subsystem base value 292, and the isolated subsystem length value 297) from the configuration storage 250 in FIG. 2B and receives access information 226 (e.g. a physical address 282). The subsystem address detector 311 detects if the physical address 282 is within a subsystem's associated memory zone as defined by the values of the current subsystem range setting configuration 273. The subsystem address detector 311 includes a masking element 322, a comparator 324, and a combiner 328. The masking element 322 masks the physical address 282 with the subsystem mask value 290. In one embodiment, the masking element 322 performs a logical AND operation. The comparator 324 compares the result of the masking operation done by the masking element 322 and the subsystem base value 292, and generates a comparison result. In other embodiments, the comparator 324 can also use the isolated length value 297 in generating the comparison result. The combiner 328 combines the comparison result with other conditions to generate a subsystem address matching signal 334. The subsystem address matching signal 334 is asserted when the physical address 282 is within a currently active subsystem's associated memory zone (75_0-75_N) as defined by the values of the current subsystem range setting configuration 273.

The access grant generator 315 combines the subsystem address matching signal 334, the execution mode word 253, and the processor isolated access signal 272 to generate the

access grant signal 274. The access grant signal 274 indicates whether a memory access request can be granted. The access grant signal 274 is asserted and a memory access is granted, if the requested memory address is outside the isolated memory area 70, OR, the processor is currently executing in isolated mode AND the address is within the currently active isolated subsystem memory range. For example, when the processor isolated access signal 272 is not asserted, the requested physical address is outside the isolated memory range and the access grant signal 274 is asserted to grant a memory access to the non-isolated memory area. Alternatively, the access grant signal 274 is asserted when the processor isolated access signal 272 is asserted and both the subsystem address matching signal 334 and the execution mode word 253 for the currently active subsystem are asserted. This indicates that an isolated access to the memory zone in the isolated memory area for the currently active subsystem is valid or allowed as configured. In one embodiment, the access grant generator 315 performs a logical AND operation of these signals. Thus, only if the physical address requested by a processor, operating in an isolated execution mode, is within the correct memory zone for the currently active subsystem will the access transaction be granted.

FIG. 4 is a flowchart illustrating a process 400 to control access to multi-memory zones of the isolated memory area for isolated execution according to one embodiment of the invention.

Upon START, the process 400 defines as an isolated memory area using the isolated setting (e.g., isolated mask value, isolated base value, and length value) (block 410). Next, the process 400 defines isolated memory zones for each subsystem in the isolated memory area using isolated subsystem range settings (e.g., subsystem mask and base values, ID's, entry attributes, and entry contexts) (block 415). Then, the process 400 receives access information from an access transaction. The access information includes a physical address as provided by the Translation Lookaside Buffer and access type (block 420). The process 400 then determines if the physical address is within the isolated memory area (block 425). If not, then the process 400 generates an access grant signal for normal execution (block 430).

If the physical address does fall within the isolated area of memory, then the process 400 determines whether the physical address falls within the memory zone of the isolated memory area for the currently active subsystem (block 435). If not, the process 400 generates a failure or fault condition (block 440). However, if the physical address does fall within the memory zone for the currently active subsystem, then the process 400 determines if the execution mode word signal is asserted for the currently active subsystem (block 445). If not, the process 400 generates a failure or fault condition (block 450). Otherwise, the process generates an access grant signal to the memory zone for the currently active subsystem for isolated execution (block 455). The process 400 is then terminated. Thus, only if the physical address requested by a processor, operating in an isolated execution mode, is within the correct memory zone for the currently active subsystem will the access transaction be granted.

While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.

What is claimed is:

1. An apparatus comprising:
a configuration storage storing configuration settings to configure an access transaction generated by a processor having a normal execution mode and an isolated execution mode, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, a memory base value, and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor, the isolated memory area being accessible to the processor in the isolated execution mode, the access transaction including access information including a physical address; and
a multi-memory zone access checking circuit coupled to the configuration storage to check the access transaction using at least one of the configuration settings and the access information and generating an access grant signal if the access transaction is valid.
2. The apparatus of claim 1 wherein each subsystem memory range setting corresponds to a memory zone for a subsystem in an isolated memory area in a memory external to the processor.
3. The apparatus of claim 1 wherein each subsystem memory range setting includes a subsystem memory base value and a subsystem memory length value, a combination of at least the subsystem base and length values to define a memory zone in the isolated memory area for a subsystem.
4. The apparatus of claim 3 wherein an ID value for each subsystem identifies each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.
5. The apparatus of claim 3 wherein the multi-memory zone access checking circuit comprises a subsystem address detector to detect if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem, the subsystem address detector generating a subsystem address matching signal.
6. The apparatus of claim 5 wherein the multi-memory zone access checking circuit further comprises an access grant generator coupled to the subsystem address detector and the processor control register, the access grant generator generating an access grant signal if both the subsystem address matching signal and the execution mode word signal are asserted.
7. An apparatus comprising:
a configuration storage storing configuration settings to configure an access transaction generated by a processor having a normal execution mode and an isolated execution mode, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, each subsystem memory range setting corresponding to a memory zone for a subsystem in an isolated memory area in a memory external to the processor and including a subsystem memory base value and a subsystem memory length value, a combination of at least the subsystem base and length values to define a memory zone in the isolated memory area for the subsystem, the

access transaction including access information including a physical address; and a multi-memory zone access checking circuit coupled to the configuration storage to check the access transaction using at least one of the configuration settings and the access information and generating an access grant signal if the access transaction is valid.

8. The apparatus of claim 7 wherein the configuration settings further include a memory base value and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor, the isolated memory area being the isolated execution mode.

9. The apparatus of claim 7 wherein an ID value for each subsystem identifies each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

10. The apparatus of claim 7 wherein the multi-memory zone access checking circuit comprises a subsystem address detector to detect if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem, the subsystem address detector generating a subsystem address matching signal.

11. The apparatus of claim 10 wherein the multi-memory zone access checking circuit further comprises an access grant generator coupled to the subsystem address detector and the processor control register, the access grant generator generating an access grant signal if both the subsystem address matching signal and the execution mode word signal are asserted.

12. A method comprising:

configuring an access transaction generated by a processor having a normal execution mode and an isolated execution mode using a configuration storage storing configuration settings, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, a memory base value, and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor that is accessible to the processor in the isolated execution mode, wherein the access transaction includes access information including a physical address;

checking the access transaction by a multi-memory zone access checking circuit using at least one of the configuration settings and the access information; and generating an access grant signal if the access transaction is valid.

13. The method of claim 12 wherein each subsystem memory range setting corresponds to a memory zone for a subsystem in an isolated memory area in a memory external to the processor.

14. The method of claim 12 wherein each subsystem memory range setting includes a subsystem memory base value and a subsystem memory length value, a combination of at the subsystem base and length values to define a memory zone in the isolated memory area for a subsystem.

15. The method of claim 14 wherein configuring the access transaction further comprises storing an ID value for each subsystem to identify each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

16. The method of claim 14 wherein checking the access transaction comprises detecting if the physical address is

within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem by a subsystem address detector, the subsystem address detector generating a subsystem address matching signal.

17. The method of claim 16 wherein generating an access grant signal if the access transaction is valid comprises generating an access grant signal by an access grant generator if both the subsystem address matching signal and the execution mode word signal are asserted.

18. A method comprising:

configuring an access transaction generated by a processor having a normal execution mode and an isolated execution mode using a configuration storage storing configuration settings, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, each subsystem memory range setting corresponding to a memory zone for a subsystem in an isolated memory area in a memory external to the processor and including a subsystem memory base value and a subsystem memory length value, a combination of at least the subsystem base and length values to define a memory zone in the isolated memory area for the subsystem, wherein the access transaction includes access information including a physical address;

checking the access transaction by a multi-memory zone access checking circuit using at least one of the configuration settings and the access information; and generating an access grant signal if the access transaction is valid.

19. The method of claim 18 wherein the configuration settings further include a memory base value and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor, the isolated memory area being accessible to the processor in the isolated execution mode.

20. The method of claim 18 wherein configuring the access transaction further comprises storing an ID value for each subsystem to identify each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

21. The method of claim 18 wherein checking the access transaction comprises detecting if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem by a subsystem address detector, the subsystem address detector generating a subsystem address matching signal.

22. The method of claim 21 wherein generating an access grant signal if the access transaction is valid comprises generating an access grant signal by an access grant generator if both the subsystem address matching signal and the execution mode word signal are asserted.

23. A computer program product comprising:

a machine readable medium having computer program code therein, the computer program product comprising:
computer readable program code for configuring an access transaction generated by a processor having a normal execution mode and an isolated execution mode using a configuration storage storing configuration settings, the configuration storage including a process control register storing an execution mode

word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, a memory base value, and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor that is accessible to the processor in the isolated execution mode, wherein the access transaction includes access information including a physical address; computer readable program code for checking the access transaction by a multi-memory zone access checking circuit using at least one of the configuration settings and the access information; and computer readable program code for generating an access grant signal if the access transaction is valid.

24. The computer program product of claim 23 wherein each subsystem memory range setting corresponds to a memory zone for a subsystem in an isolated memory area in a memory external to the processor.

25. The computer program product of claim 23 wherein each subsystem memory range setting includes a subsystem memory base value and a subsystem memory length value, a combination of at least the subsystem base and length values to define a memory zone in the isolated memory area for a subsystem.

26. The computer program product of claim 25 wherein the computer readable code for configuring the access transaction further comprises computer readable code storing an ID value for each subsystem to identify each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

27. The computer program product of claim 25 wherein the computer readable code for checking the access transaction comprises computer readable code for detecting if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem by a subsystem address detector, the subsystem address detector generating a subsystem address matching signal.

28. The computer program product of claim 27 wherein the computer readable code for generating an access grant signal if the access transaction is valid comprises computer readable code for generating an access grant signal by an access grant generator if both the subsystem address matching signal and the execution mode word signal are asserted.

29. A computer program product comprising:

a machine readable medium having computer program code therein, the computer program product comprising:
50
computer readable program code for configuring an access transaction generated by a processor having a normal execution mode and an isolated execution mode using a configuration storage storing configuration settings, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, each subsystem memory range setting corresponding to a memory zone for a subsystem in an isolated memory area in a memory external to the processor and including a subsystem memory base value and a subsystem memory length value, a combination of at least the subsystem base and length values to define

a memory zone in the isolated memory area for the subsystem, wherein the access transaction includes access information including a physical address; computer readable program code for checking the access transaction by a multi-memory zone access checking circuit using at least one of the configuration settings and the access information; and computer readable program code for generating an access grant signal if the access transaction is valid.

30. The computer program product of claim 29 wherein the configuration settings further include a memory base value and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor, the isolated memory area being accessible to the processor in the isolated execution mode.

31. The computer program product of claim 29 wherein the computer readable code for configuring the access transaction further comprises computer readable code for storing an ID value for each subsystem to identify each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

32. The computer program product of claim 29 wherein the computer readable code for checking the access transaction comprises computer readable code for detecting if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem by a subsystem address detector, the subsystem address detector generating a subsystem address matching signal.

33. The computer program product of claim 32 wherein the computer readable code for generating an access grant signal if the access transaction is valid comprises computer readable code for generating an access grant signal by an access grant generator if both the subsystem address matching signal and the execution mode word signal are asserted.

34. A system comprising:

a chipset;
a memory coupled to the chipset having an isolated memory area;
a processor coupled to the chipset and the memory having an access manager, the processor having a normal execution mode and an isolated execution mode, the processor generating an access transaction having access information, the access manager comprising:
a configuration storage storing configuration settings to configure an access transaction generated by a processor having a normal execution mode and an isolated execution mode, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the isolated execution mode, the configuration settings including a plurality of subsystem memory range settings, a memory base value, and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor, the isolated memory area being accessible to the processor in the isolated execution mode, the access transaction including access information including a physical address; and
a multi-memory zone access checking circuit coupled to the configuration storage to check the access transaction using at least one of the configuration settings and the access information and generating an access grant signal if the access transaction is valid.

35. The system of claim 34 wherein each subsystem memory range setting corresponds to a memory zone for a subsystem in an isolated memory area in a memory external to the processor.

36. The system of claim 34 wherein each subsystem memory range setting includes a subsystem memory base value and a subsystem memory length value, a combination of at least the base and length values to define a memory zone in the isolated memory area for a subsystem.

37. The system of claim 36 wherein an ID value for each subsystem identifies each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

38. The system of claim 36 wherein the multi-memory zone access checking circuit comprises a subsystem address detector to detect if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem, the subsystem address detector generating a subsystem address matching signal.

39. The system of claim 38 wherein the multi-memory zone access checking circuit further comprises an access grant generator coupled to the subsystem address detector and the processor control register, the access grant generator generating an access grant signal if both the subsystem address matching signal and the execution mode word signal are asserted.

40. A system comprising:

- a chipset;
- a memory coupled to the chipset having an isolated memory area;
- a processor coupled to the chipset and the memory having an access manager, the processor having a normal execution mode and an isolated execution mode, the processor generating an access transaction having access information, the access manager comprising:
 - a configuration storage storing configuration settings to configure an access transaction generated by a processor having a normal execution mode and an isolated execution mode, the configuration storage including a process control register storing an execution mode word that is asserted as an execution mode signal when the processor is configured in the iso-

lated execution mode, the configuration settings including a plurality of subsystem memory range settings, each subsystem memory range setting corresponding to a memory zone for a subsystem in an isolated memory area in a memory external to the processor and including a subsystem memory base value and a subsystem memory length value, a combination of at least the subsystem base and length values to define a memory zone in the isolated memory area for the subsystem, the access transaction including access information including a physical address; and

a multi-memory zone access checking circuit coupled to the configuration storage to check the access transaction using at least one of the configuration settings and the access information and generating an access grant signal if the access transaction is valid.

41. The system of claim 40 wherein the configuration settings further include a memory base value and a memory length value, a combination of at least the base and length values to define an isolated memory area in a memory external to the processor, the isolated memory area being accessible to the processor in the isolated execution mode.

42. The system of claim 40 wherein an ID value for each subsystem identifies each subsystem and the subsystem's associated memory zone as defined by the subsystem memory range setting.

43. The system of claim 40 wherein the multi-memory zone access checking circuit comprises a subsystem address detector to detect if the physical address is within a currently active subsystem's associated memory zone as defined by the subsystem memory range setting for the subsystem, the subsystem address detector generating a subsystem address matching signal.

44. The system of claim 43 wherein the multi-memory zone access checking circuit further comprises an access grant generator coupled to the subsystem address detector and the processor control register, the access grant generator generating an access grant signal if both the subsystem address matching signal and the execution mode word signal are asserted.

* * * * *



US006507904B1

(12) **United States Patent**
Ellison et al.

(10) **Patent No.:** US 6,507,904 B1
(45) **Date of Patent:** Jan. 14, 2003

(54) **EXECUTING ISOLATED MODE INSTRUCTIONS IN A SECURE SYSTEM RUNNING IN PRIVILEGE RINGS**

(75) Inventors: Carl M. Ellison, Portland; Roger A. Golliver, Beaverton, both of OR (US); Howard C. Herbert, Phoenix, AZ (US); Derrick C. Lin, Foster City, CA (US); Francis X. McKeen; Gilbert Nelger, both of Portland, OR (US); Ken Reneris, Wilbraham, MA (US); James A. Sutton, Portland, OR (US); Shreekant S. Thakkar, Portland, OR (US); Millind Mittal, Palo Alto, CA (US)

(73) Assignee: **Intel Corporation**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: 09/541,477

(22) Filed: Mar. 31, 2000

(51) Int. Cl.⁷ G06F 9/44

(52) U.S. Cl. 712/229; 711/152; 709/100

(58) Field of Search 712/229; 713/164, 713/166; 711/152; 709/100

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,037,214 A	7/1977	Birney et al.	711/216
4,278,837 A	7/1981	Best	713/190
4,366,537 A	12/1982	Heller et al.	711/164
4,521,852 A	6/1985	Guttag	711/163
5,022,077 A	6/1991	Bealkowski et al.	711/163
5,075,842 A	12/1991	Lai	711/152
5,255,379 A *	10/1993	Melo	711/202
5,293,424 A	3/1994	Holtley et al.	713/193
5,421,006 A	5/1995	Jablon et al.	714/36
5,459,869 A	10/1995	Spiro	709/324
5,473,692 A	12/1995	Davis	705/59

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	0600112	8/1994
EP	0930567 A	7/1999
EP	1146715	10/2001
WO	WO 9834365 A	8/1998
WO	WO9844402	10/1998
WO	WO9905600	2/1999
WO	WO9909482	2/1999
WO	WO9957863	11/1999
WO	WO0062232	10/2000

OTHER PUBLICATIONS

J. Heinrich: "MIPS R4000 Microprocessor User's Manual," Apr. 1, 1993, MIPS, MT. View, XP002184449, pp. 61-97. "M68040 User's Manual", 1993, Motorola INC., p. 1-5-p. 1-9, p. 1-13-p. 1-20, p. 2-1-p. 2-3, p. 4-1, p. 8-9-p. 8-11. "Intel 386 DX Microprocessor 32-Bit CMOS Microprocessor With Integrated Memory Management", Dec. 31, 1995, Intel Inc., pp. 32-56; figure 4-14.

Joe Heinrich: "MIPS R4000 Microprocessor User's Manual", 1994, MIPS Technology Inc., Mountain View, CA, pp. 67-79.

Berg C: "How Do I Create a Signed Applet?", Dr. Dobb's Journal, M&T Publ., Redwood City, CA, US, vol. 22, No. 8, 8 '97, p. 109-111, 122.

Gong L et al: "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", Proceedings of the Usenix Symposium on Internet Technologies and Systems, Monterey, CA 13 '97, pp. 103-112.

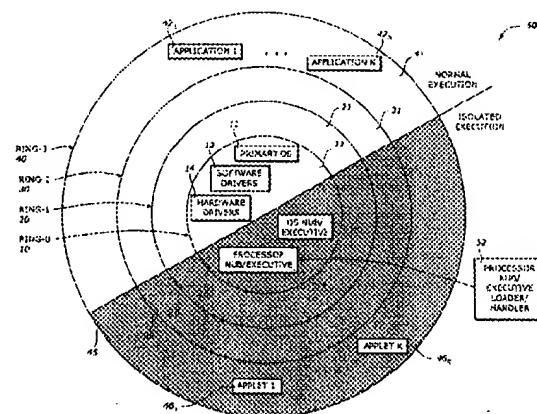
Primary Examiner—Kenneth S. Kim

(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

A technique is provided to execute isolated instructions according to an embodiment of the present invention. An execution unit executes an isolated instruction in a processor operating in a platform. The processor is configured in one of a normal execution mode and an isolated execution mode. A parameter storage containing at least one parameter to support execution of the isolated instruction when the processor is configured in the isolated execution mode.

56 Claims, 13 Drawing Sheets



U.S. PATENT DOCUMENTS

5,479,509 A	12/1995	Ugon	713/176	5,937,063 A	8/1999	Davis	713/187
5,568,552 A	10/1996	Davis	705/59	5,978,481 A	11/1999	Ganesan et al.	380/266
5,615,263 A *	3/1997	Takahashi	713/164	5,987,557 A	11/1999	Ebrahim	710/200
5,657,445 A	8/1997	Pearce	713/200	6,014,745 A	1/2000	Ashe	713/193
5,717,903 A *	2/1998	Bonola	703/24	6,058,478 A	5/2000	Davis	713/191
5,729,760 A	3/1998	Poisner	710/3	6,175,925 B1	1/2001	Nardone et al.	713/200
5,764,969 A *	6/1998	Kahle et al.	712/228	6,178,509 B1	1/2001	Nardone et al.	713/200
5,796,845 A	8/1998	Serikawa et al.	381/18	6,205,550 B1	3/2001	Nardone et al.	713/200
5,805,712 A	9/1998	Davis	713/173	6,282,651 B1	8/2001	Ashe	713/193
5,835,594 A	11/1998	Albrecht et al.	713/187	6,282,657 B1	8/2001	Kaplan et al.	713/201
5,844,986 A	12/1998	Davis	713/187	2001/0037450 A1	11/2001	Metlitski et al.	713/152
5,890,189 A	3/1999	Nozue et al.	711/100				

* cited by examiner

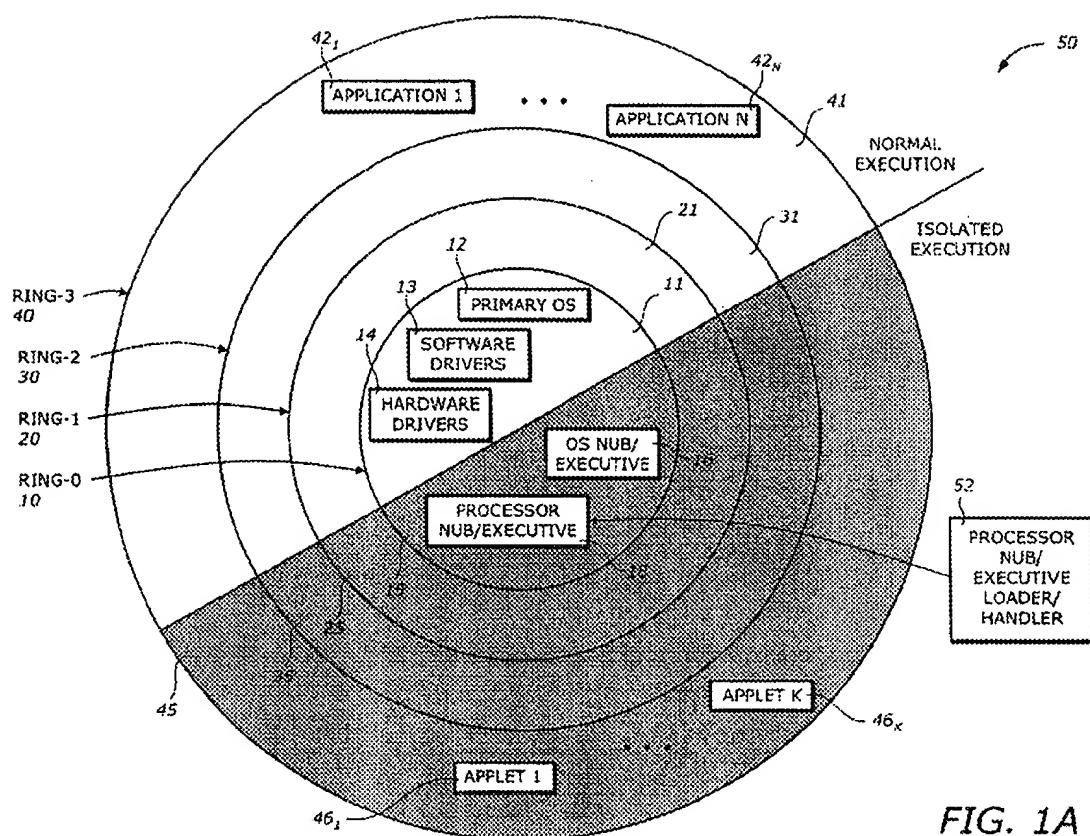


FIG. 1A

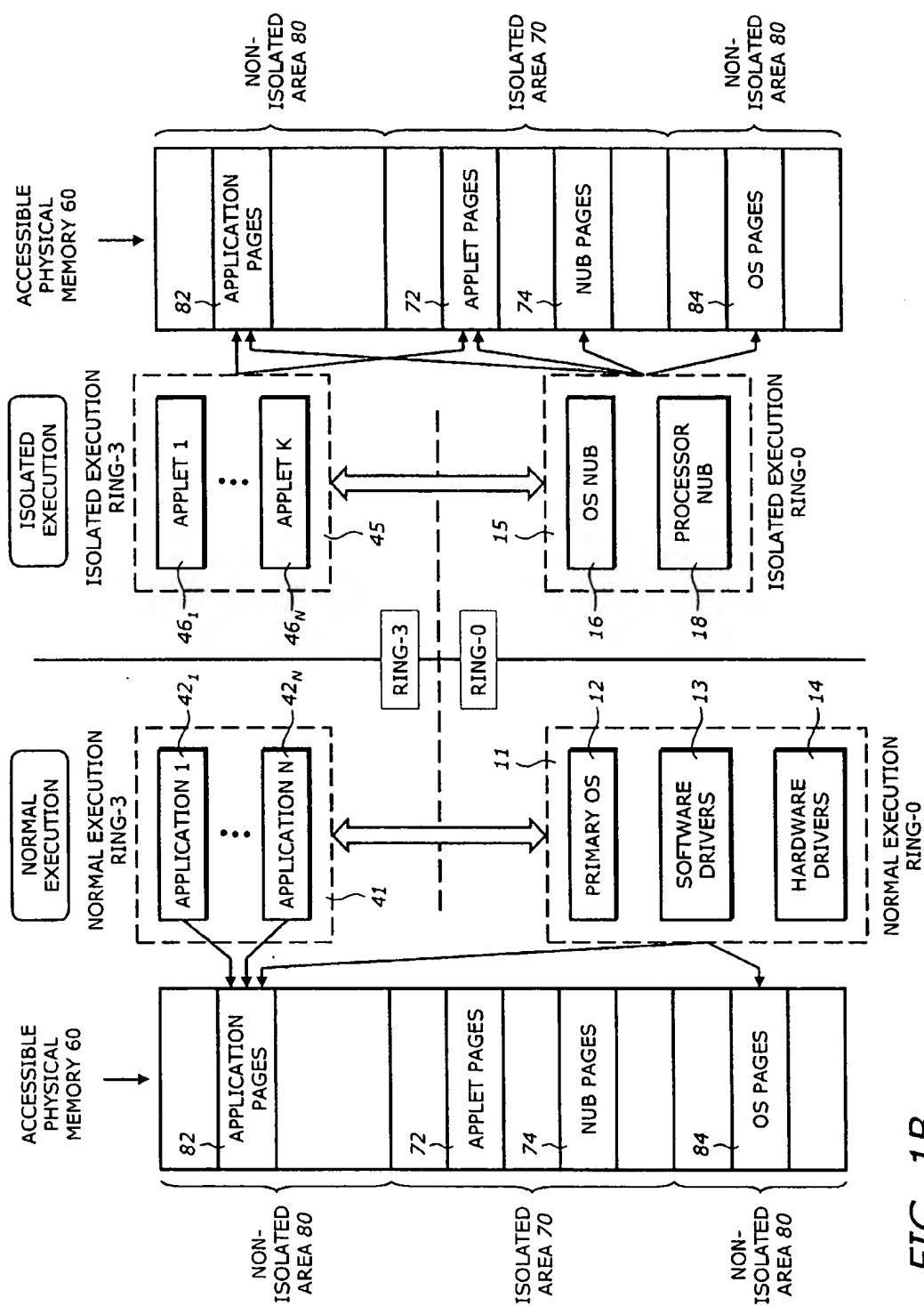


FIG. 1B

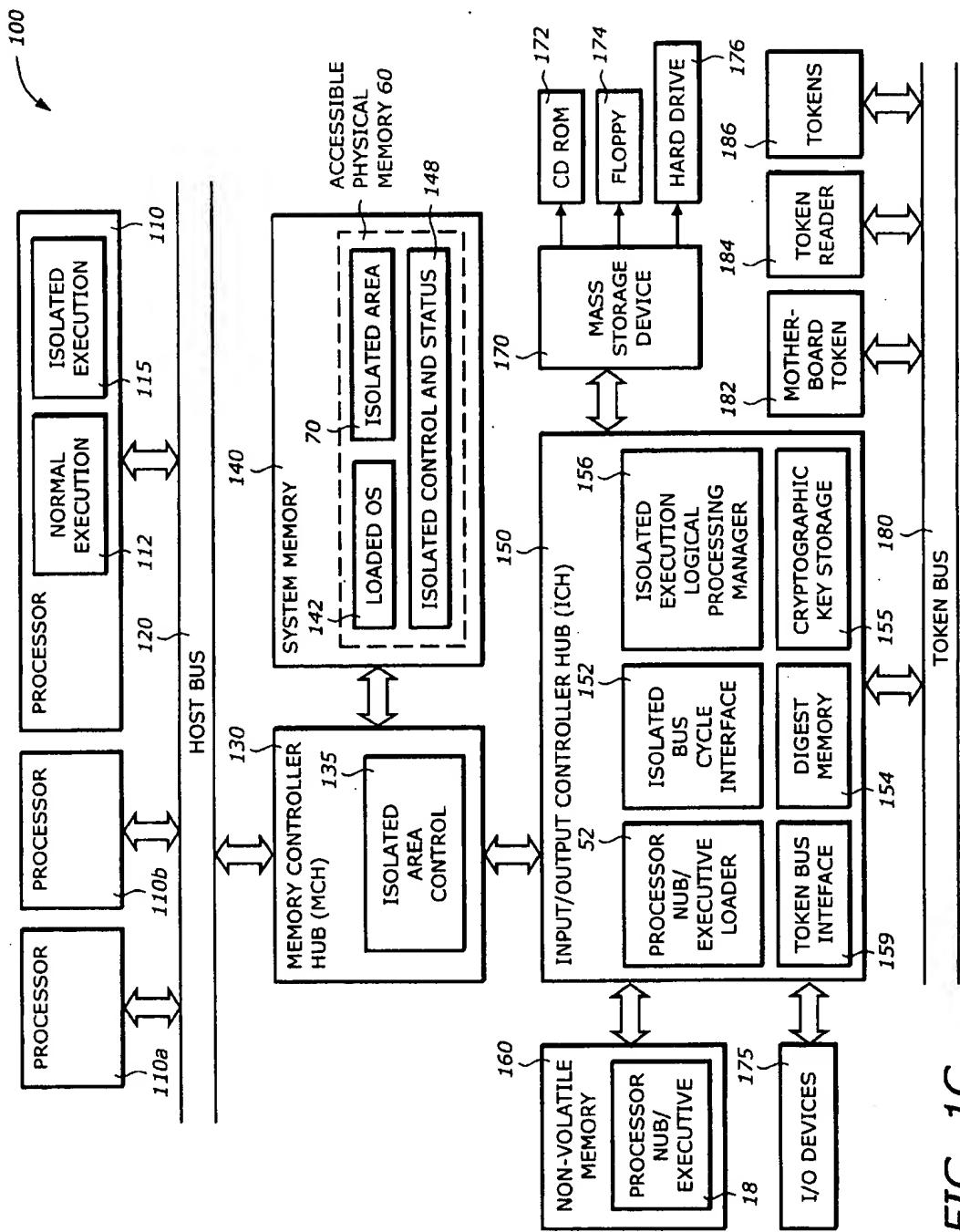


FIG. 1C

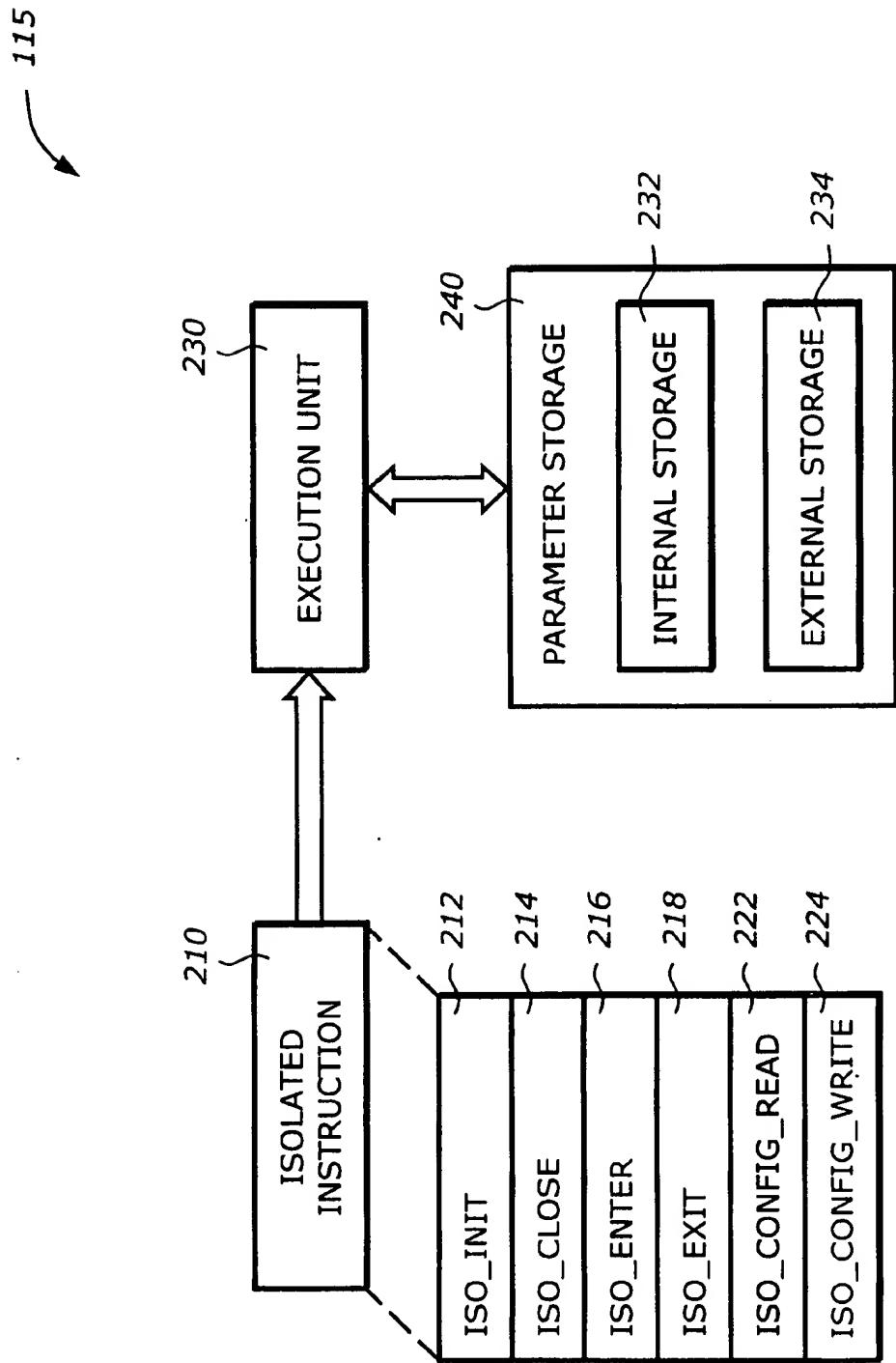


FIG. 2

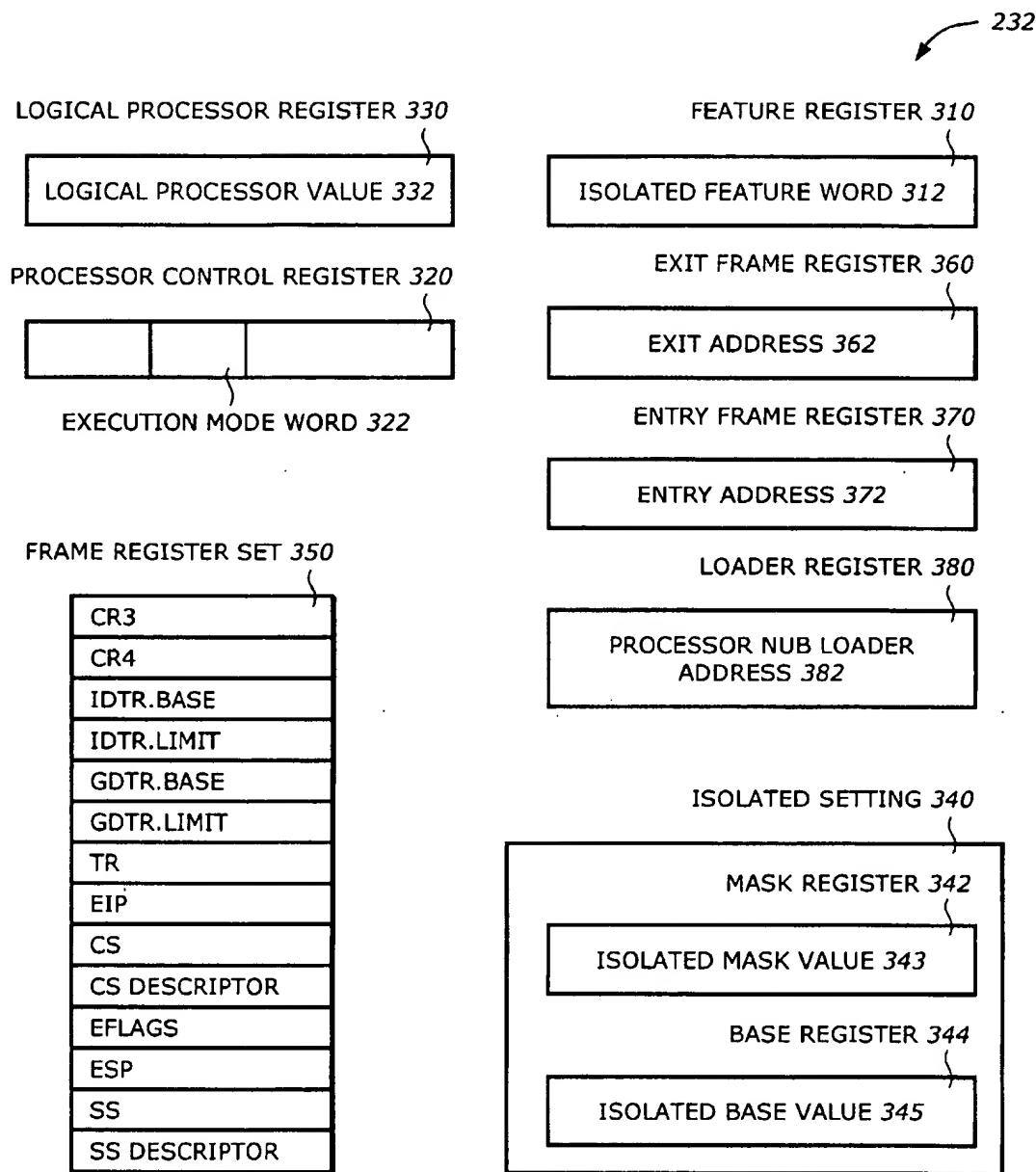


FIG. 3

234

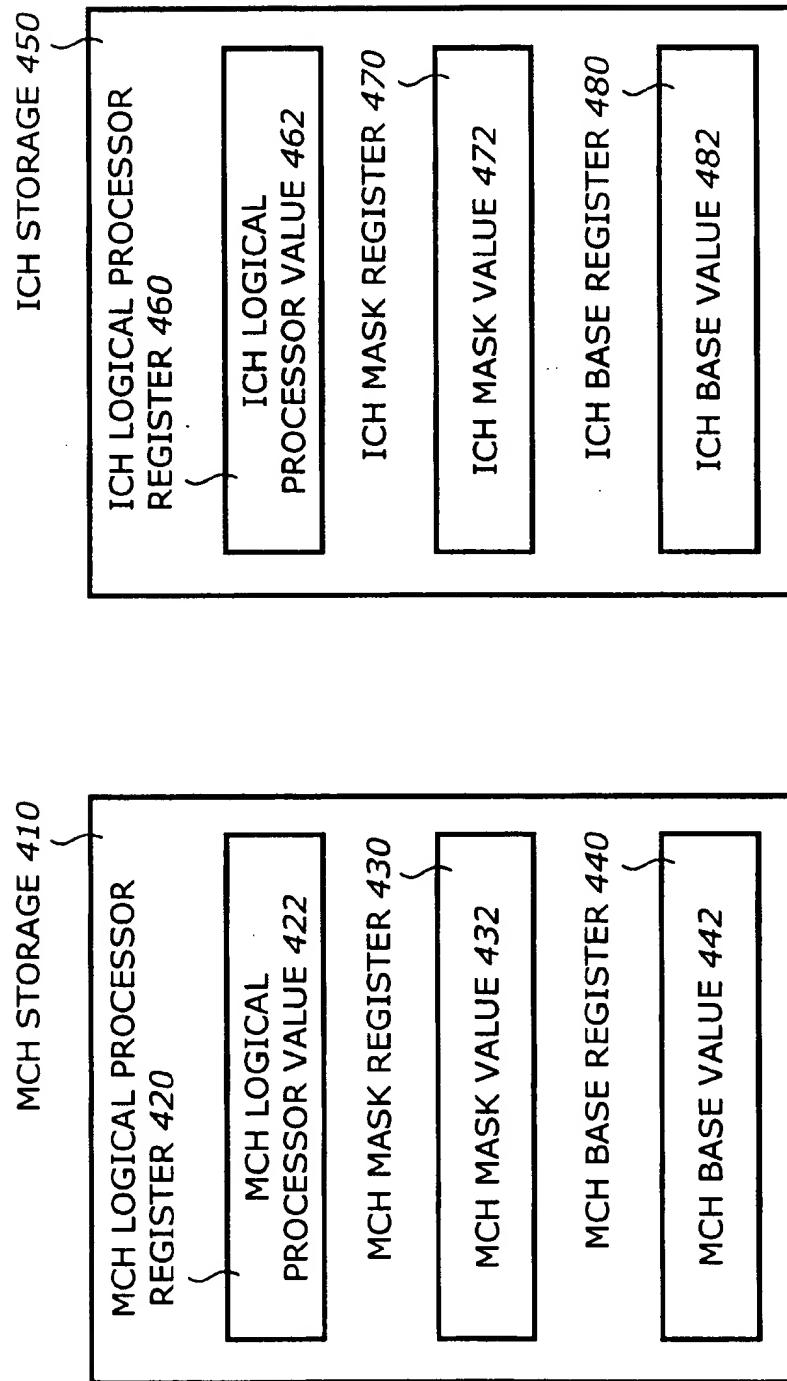


FIG. 4

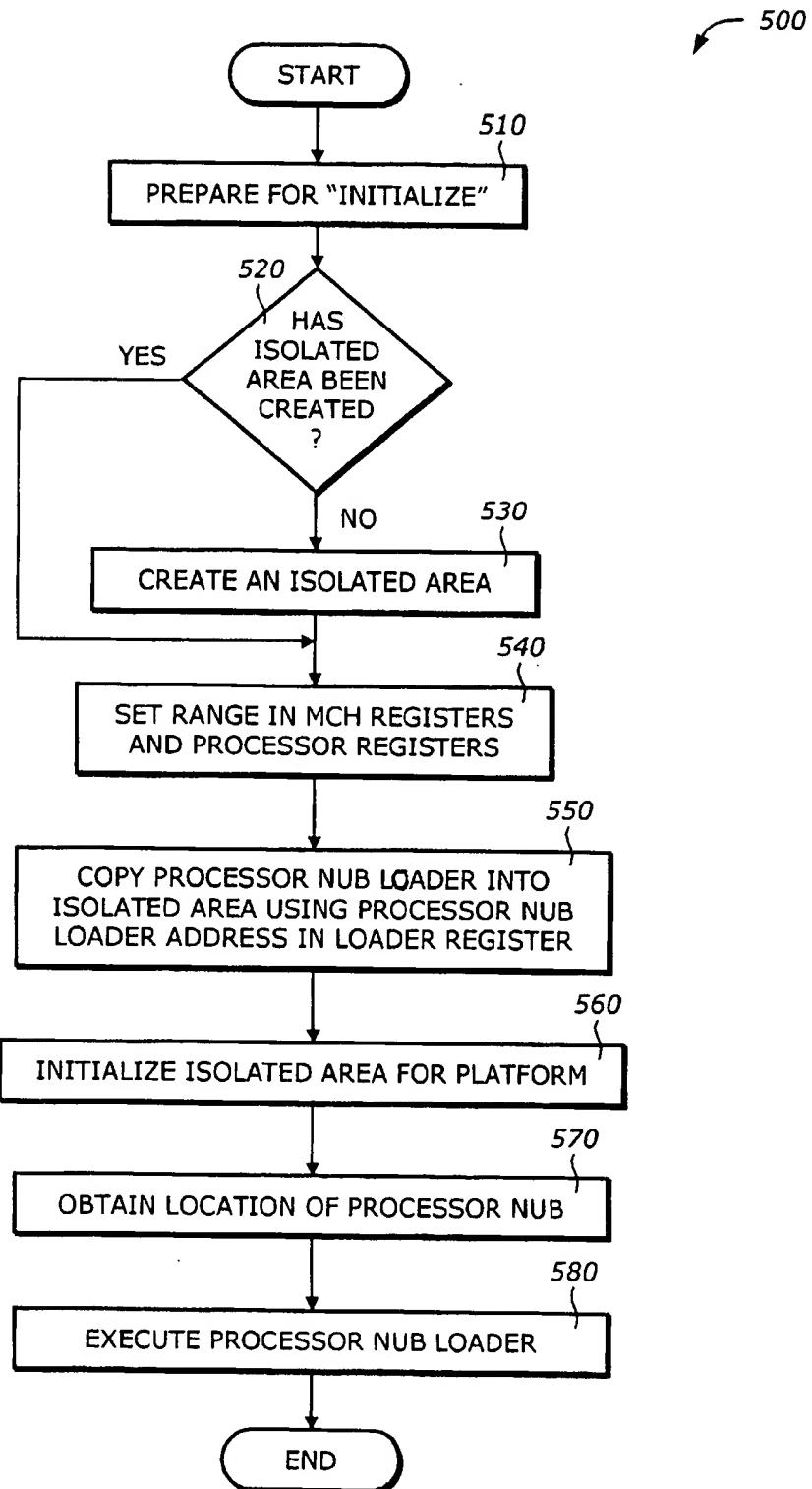


FIG. 5

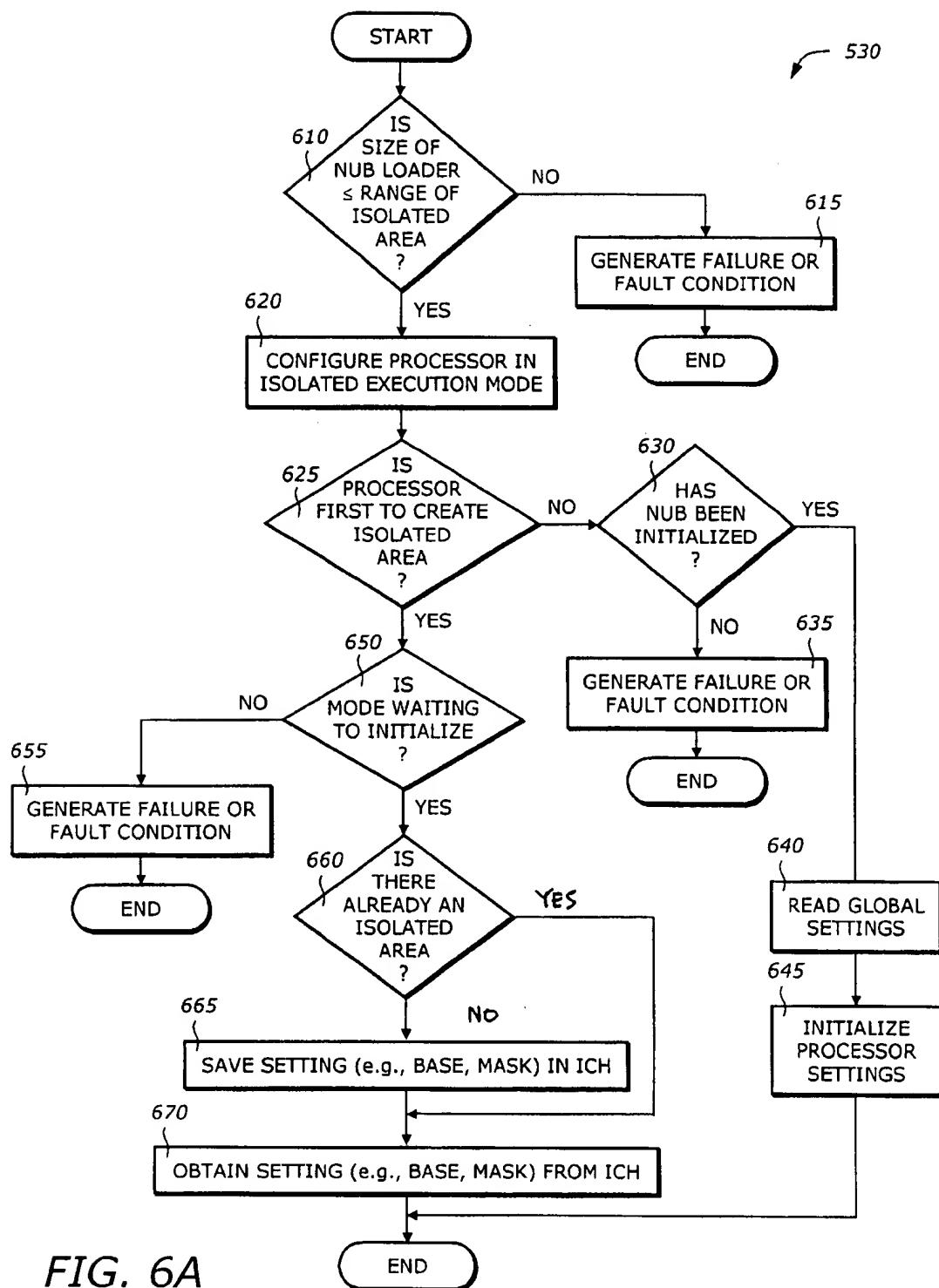


FIG. 6A

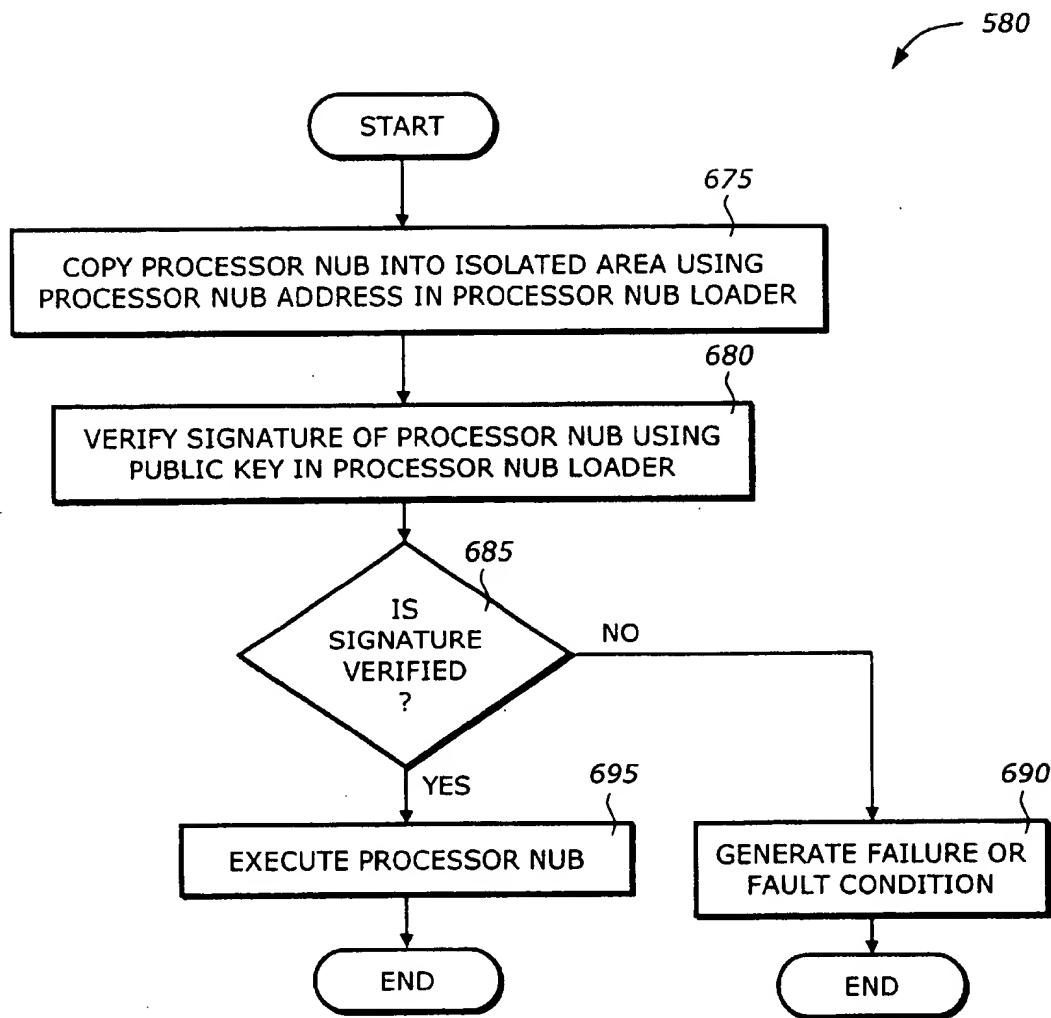


FIG. 6B

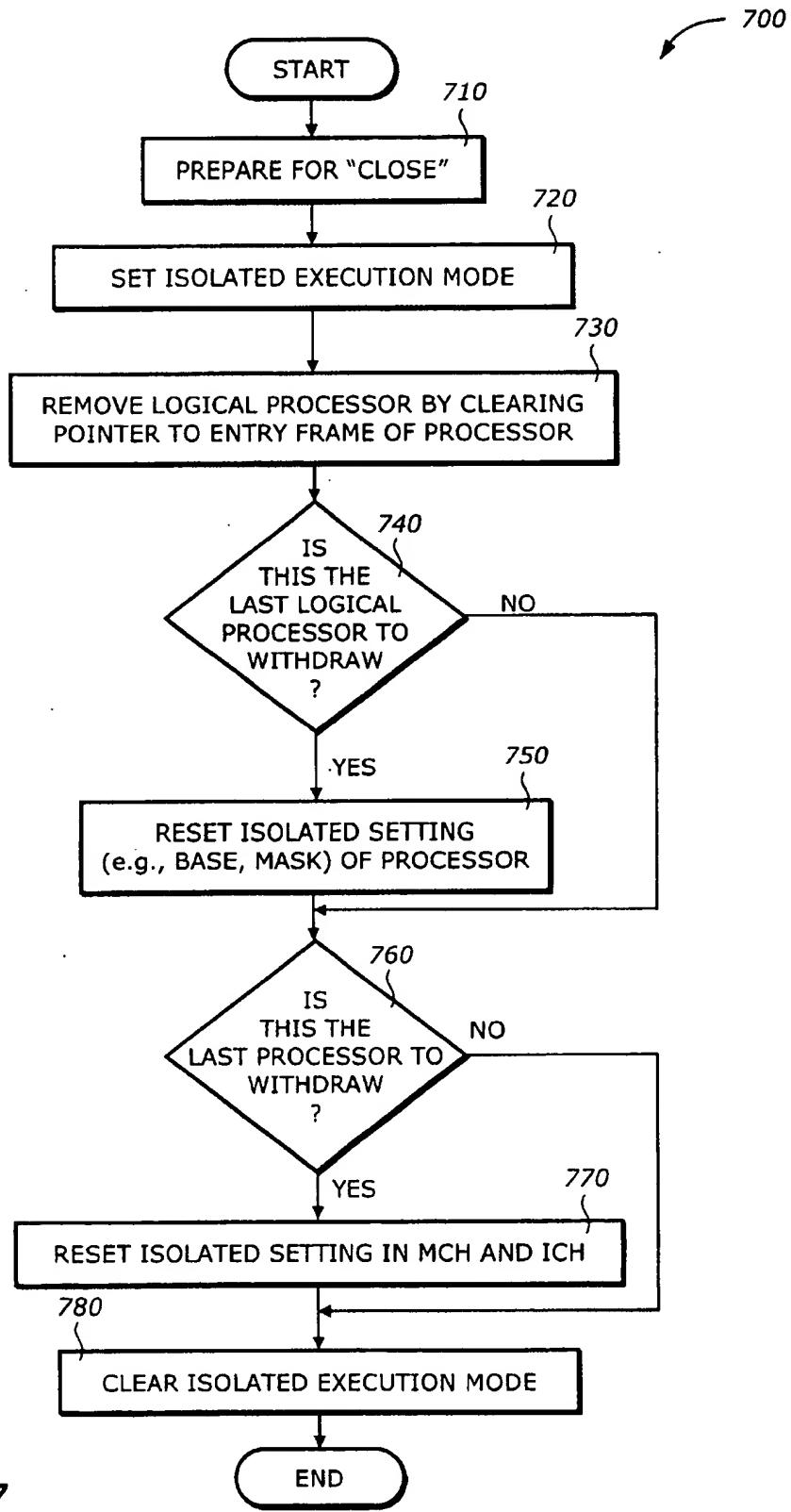


FIG. 7

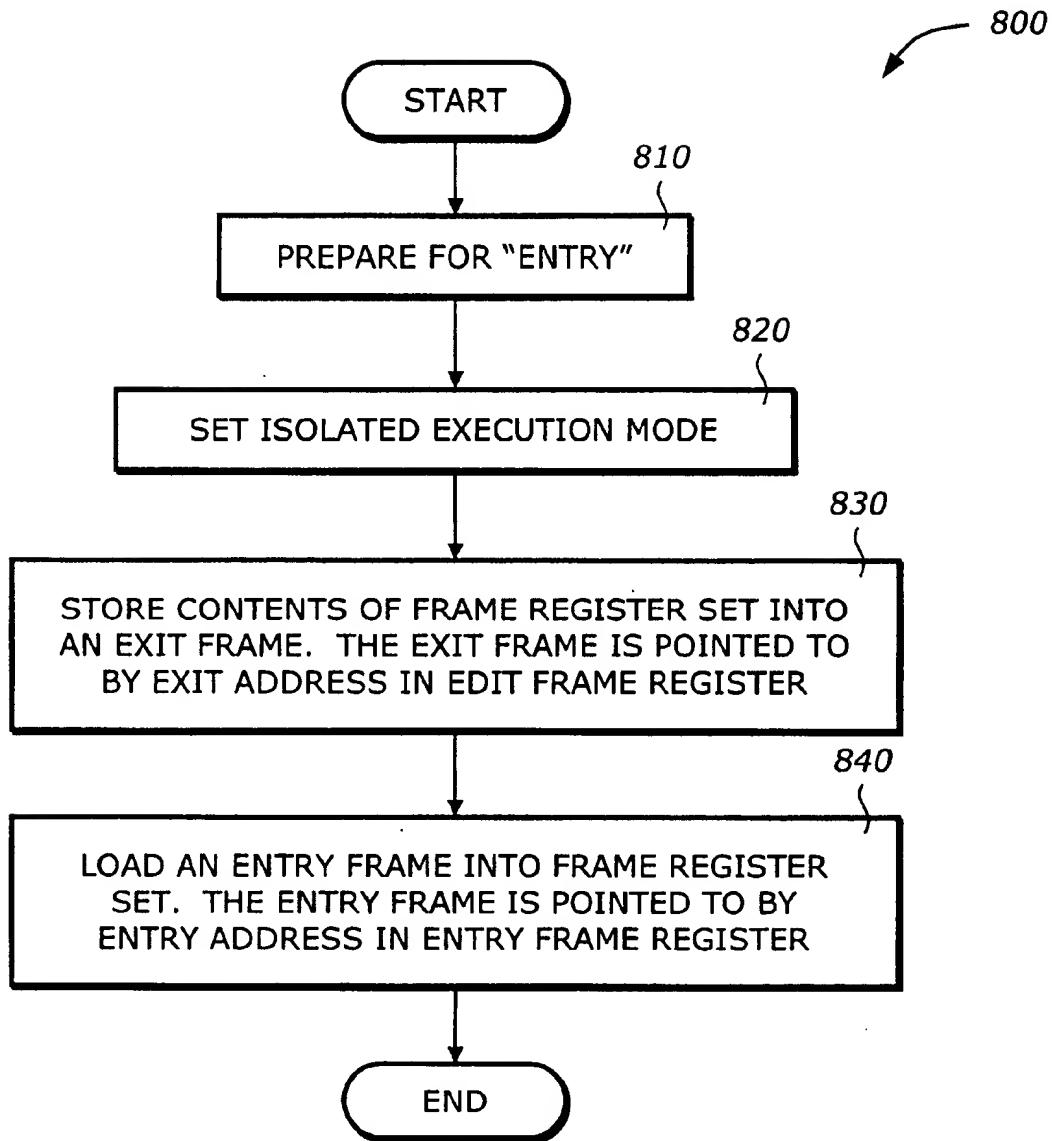


FIG. 8A

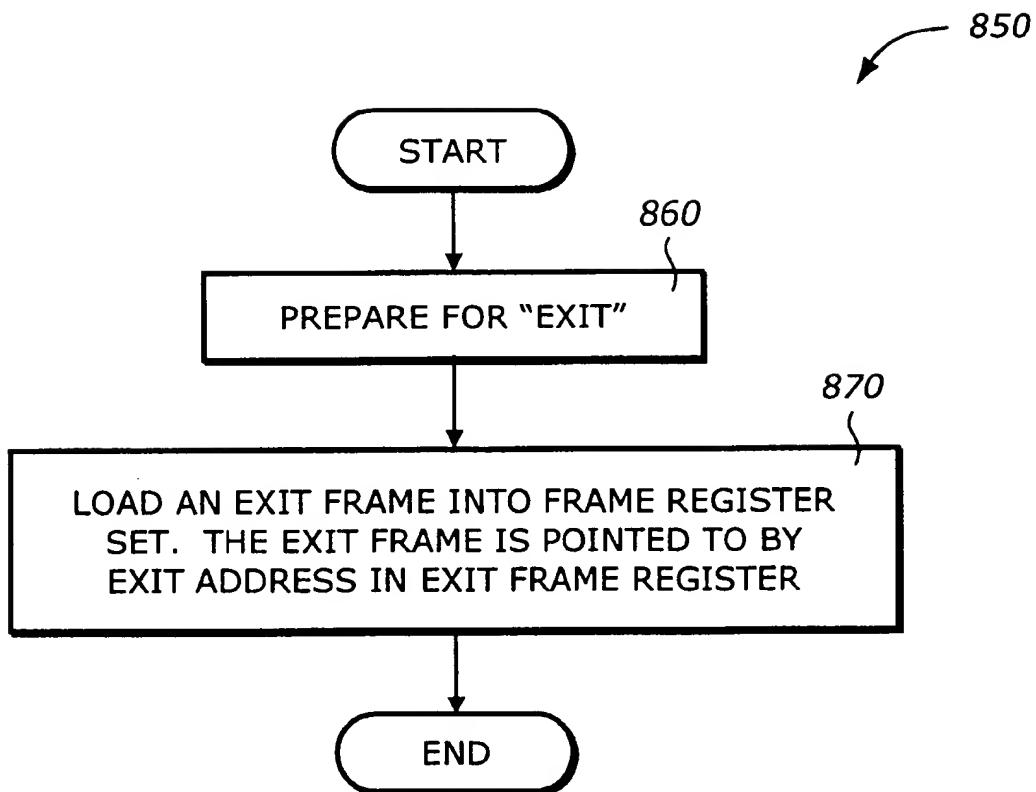


FIG. 8B

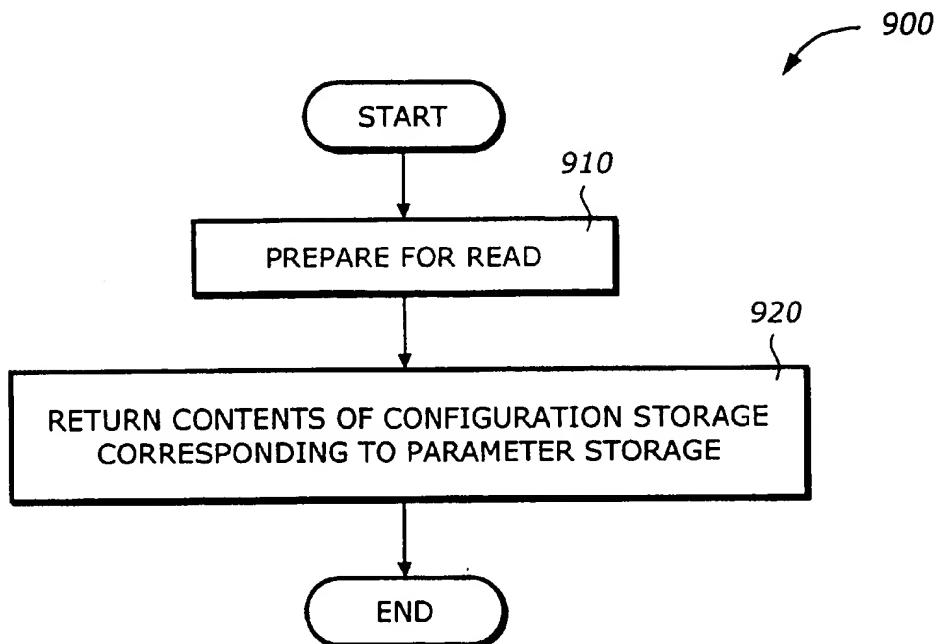


FIG. 9A

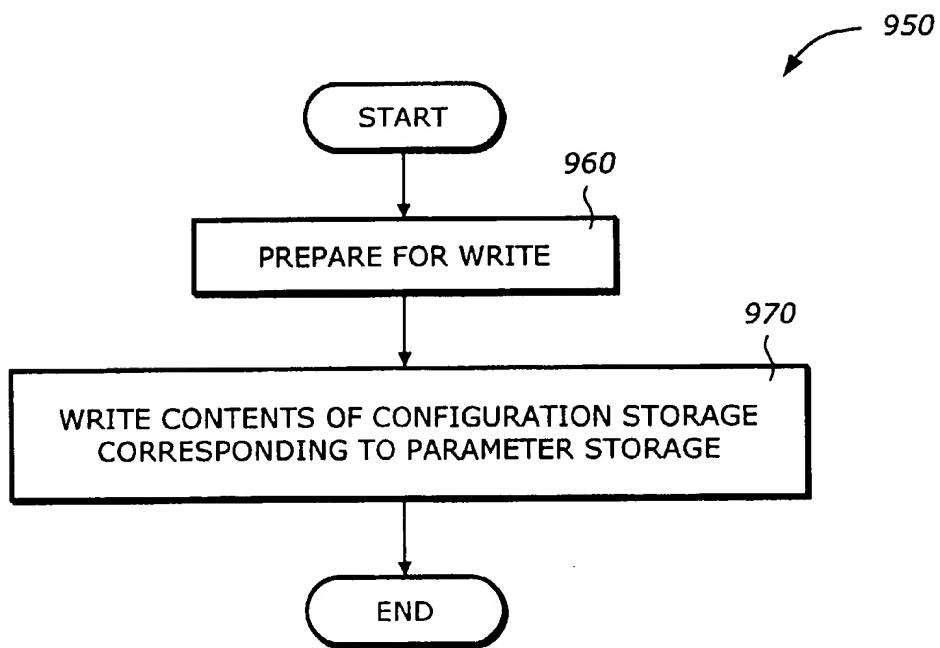


FIG. 9B

**EXECUTING ISOLATED MODE
INSTRUCTIONS IN A SECURE SYSTEM
RUNNING IN PRIVILEGE RINGS**

BACKGROUND

1. Field of the Invention

This invention relates to microprocessors. In particular, the invention relates to processor security.

2. Description of Related Art

Advances in microprocessor and communication technologies have opened up many opportunities for applications that go beyond the traditional ways of doing business. Electronic commerce (E-commerce) and business-to-business (B2B) transactions are now becoming popular, reaching the global markets at a fast rate. Unfortunately, while modern microprocessor systems provide users convenient and efficient methods of doing business, communicating and transacting, they are also vulnerable to unscrupulous attacks. Examples of these attacks include virus, intrusion, security breach, and tampering, to name a few. Computer security, therefore, is becoming more and more important to protect the integrity of the computer systems and increase the trust of users.

Threats caused by unscrupulous attacks may be in a number of forms. Attacks may be remote without requiring physical accesses. An invasive remote-launched attack by hackers may disrupt the normal operation of a system connected to thousands or even millions of users. A virus program may corrupt code and/or data of a single-user platform.

Existing techniques to protect against attacks have a number of drawbacks. Anti-virus programs can only scan and detect known viruses. Most anti-virus programs use a weak policy in which a file or program is assumed good until proved bad. For many security applications, this weak policy may not be appropriate. In addition, most anti-virus programs are used locally where they are resident in the platform. This may not be suitable in a group work environment. Security co-processors or smart cards using cryptographic or other security techniques have limitations in speed performance, memory capacity, and flexibility. Redesigning operating systems creates software compatibility issues and causes tremendous investment in development efforts.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

FIG. 1A is a diagram illustrating an logical operating architecture according to one embodiment of the invention.

FIG. 1B is a diagram illustrating accessibility of various elements in the operating system and the processor according to one embodiment of the invention.

FIG. 1C is a diagram illustrating a computer system in which one embodiment of the invention can be practiced.

FIG. 2 is a diagram illustrating an architecture for isolated instructions according to one embodiment of the invention.

FIG. 3 is a diagram illustrating an internal storage according to one embodiment of the invention.

FIG. 4 is a diagram illustrating an external storage according to one embodiment of the invention.

FIG. 5 is a flowchart illustrating a process to execute the iso_init instruction according to one embodiment of the invention.

FIG. 6A is a flowchart illustrating a process to execute the creating function in the iso_init instruction according to one embodiment of the invention.

FIG. 6B is a flowchart illustrating a process to execute the processor nub loader function in the iso_init instruction according to one embodiment of the invention.

FIG. 7 is a flowchart illustrating a process to execute the iso_close instruction according to one embodiment of the invention.

FIG. 8A is a flowchart illustrating a process to execute the iso_enter instruction according to one embodiment of the invention.

FIG. 8B is a flowchart illustrating a process to execute the iso_exit instruction according to one embodiment of the invention.

FIG. 9A is a flowchart illustrating a process to execute the iso_config_read instruction according to one embodiment of the invention.

FIG. 9B is a flowchart illustrating a process to execute the iso_config_write instruction according to one embodiment of the invention.

DESCRIPTION

In an embodiment of the present invention, a technique is provided to execute isolated instructions. An execution unit executes an isolated instruction in a processor operating in a platform. The processor is configured in one of a normal execution mode and an isolated execution mode. A parameter storage contains at least one parameter to support execution of the isolated instruction when the processor is configured in the isolated execution mode.

In one embodiment, the isolated instruction is one of an isolated initialize (iso_init) instruction, an isolated close (iso_close) instruction, an isolated enter (iso_enter) instruction, an isolated exit (iso_exit) instruction, an isolated configuration read (iso_config_read), and an isolated configuration write (iso_config_write) instruction. The parameter is one of an isolated feature word, an execution mode word, a logical processor value, an isolated setting including a mask value and a base value, an exit physical address, an entry physical address, and a processor nub loader physical address.

The parameter storage is one of an internal storage and an external storage. The internal storage includes a control register to store the execution mode word, a logical processor register, a mask register to store the mask value, a base register to store the base value, a frame register set, an exit frame register to store the exit address, an entry frame register to store the entry address, and a loader register to store the processor nub loader address. The external storage includes a memory controller hub (MCH) storage and an input/output controller hub (ICH) storage. The execution mode word configures the processor in the isolated execution mode.

The iso_init instruction, when executed, causes the processor to create an isolated area defined by the base value and the mask value in a memory if the isolated area has not been created, set a range of the isolated area in the MCH register, copy a processor nub loader into the isolated area using the processor nub loader address in the loader register, and execute the processor nub loader. The iso_init instruction, when executed, further causes the processor to initialize the isolated area for the platform, and obtain a location of a processor nub. The iso_init instruction causing the processor to execute the processor nub loader causes the

processor to copy the processor nub into the isolated area, verify a signature of the processor nub using a public key contained in the processor nub loader, and execute the processor nub if the signature is verified. The iso_init instruction, therefore, ensures that the state of the environment or platform is clean upon creation of the isolated area.

The iso_enter instruction, when executed, causes the processor to store contents of the frame register set in an exit frame, the exit frame being pointed to by the exit address in the exit frame register, and load an entry frame into the frame register set, the entry frame being pointed to by the entry address in the entry frame register. The iso_exit instruction, when executed, causes the processor to load an exit frame into the frame register set. The exit frame is pointed by the exit address in the exit frame register. The iso_close instruction, when executed, causes the processor to reset the isolated setting in the processor if a corresponding logical processor is last to withdraw, and reset the isolated setting in the MCH and the ICH if the processor is last to withdraw. The iso_config_read instruction, when executed, causes the processor to return contents of a configuration storage corresponding to the parameter storage. The iso_config_write instruction, when executed, causes the processor to write contents of a configuration storage corresponding to the parameter storage.

The following instructions are atomic, non-interruptible, to assure the integrity of the environment and/or platform: iso_enter, iso_exit, iso_close, iso_config_read, iso_config_write. The iso_enter instruction, when executed, causes the processor to store contents of the frame register set in an exit frame, the exit frame being pointed to by the exit address in the exit frame register, and load an entry frame into the frame register set, the entry frame being pointed to by the entry address in the entry frame register. The iso_exit instruction, when executed, causes the processor to load an exit frame into the frame register set. The exit frame is pointed by the exit address in the exit frame register. The iso_close instruction, when executed, causes the processor to reset the isolated setting in the processor if a corresponding logical processor is last to withdraw, and reset the isolated setting in the MCH and the ICH if the processor is last to withdraw. The iso_config_read instruction, when executed, causes the processor to return contents of a configuration storage corresponding to the parameter storage. The iso_config_write instruction, when executed, causes the processor to write contents of a configuration storage corresponding to the parameter storage.

In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well-known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention.

Architecture Overview

One principle for providing security in a computer system or platform is the concept of an isolated execution architecture. The isolated execution architecture includes logical and physical definitions of hardware and software components that interact directly or indirectly with an operating system of the computer system or platform. An operating system and the processor may have several levels of hierarchy, referred to as rings, corresponding to various operational modes. A ring is a logical division of hardware and software

components that are designed to perform dedicated tasks within the operating system. The division is typically based on the degree or level of privilege, namely, the ability to make changes to the platform. For example, a ring-0 is the innermost ring, being at the highest level of the hierarchy. Ring-0 encompasses the most critical, privileged components. In addition, modules in Ring-0 can also access to lesser privileged data, but not vice versa. Ring-3 is the outermost ring, being at the lowest level of the hierarchy. Ring-3 typically encompasses users or applications level and has the least privilege. Ring-1 and ring-2 represent the intermediate rings with decreasing levels of privilege.

FIG. 1A is a diagram illustrating a logical operating architecture 50 according to one embodiment of the invention. The logical operating architecture 50 is an abstraction of the components of an operating system and the processor. The logical operating architecture 50 includes ring-0 10, ring-1 20, ring-2 30, ring-3 40, and a processor nub loader 52. The processor nub loader 52 is an instance of a processor executive (PE) handler. The PE handler is used to handle and/or manage a processor executive (PE) as will be discussed later. The logical operating architecture 50 has two modes of operation: normal execution mode and isolated execution mode. Each ring in the logical operating architecture 50 can operate in both modes. The processor nub loader 52 operates only in the isolated execution mode.

Ring-0 10 includes two portions: a normal execution Ring-0 11 and an isolated execution Ring-0 15. The normal execution Ring-0 11 includes software modules that are critical for the operating system, usually referred to as kernel. These software modules include primary operating system (e.g., kernel) 12, software drivers 13, and hardware drivers 14. The isolated execution Ring-0 15 includes an operating system (OS) nub 16 and a processor nub 18. The OS nub 16 and the processor nub 18 are instances of an OS executive (OSE) and processor executive (PE), respectively. The OSE and the PE are part of executive entities that operate in a secure environment associated with the isolated area 70 and the isolated execution mode. The processor nub loader 52 is a protected bootstrap loader code held within a chipset in the system and is responsible for loading the processor nub 18 from the processor or chipset into an isolated area as will be explained later.

Similarly, ring-1 20, ring-2 30, and ring-3 40 include normal execution ring-1 21, ring-2 31, ring-3 41, and isolated execution ring-1 25, ring-2 35, and ring-3 45, respectively. In particular, normal execution ring-3 includes N applications 42₁ to 42_N and isolated execution ring-3 includes K applets 46₁ to 46_K.

One concept of the isolated execution architecture is the creation of an isolated region in the system memory, referred to as an isolated area, which is protected by both the processor and chipset in the computer system. The isolated region may also be in cache memory, protected by a translation look aside (TLB) access check. Access to this isolated region is permitted only from a front side bus (FSB) of the processor, using special bus (e.g., memory read and write) cycles, referred to as isolated read and write cycles. The special bus cycles are also used for snooping. The isolated read and write cycles are issued by the processor executing in an isolated execution mode. The isolated execution mode is initialized using a privileged instruction in the processor, combined with the processor nub loader 52. The processor nub loader 52 verifies and loads a ring-0 nub software module (e.g., processor nub 18) into the isolated area. The processor nub 18 provides hardware-related services for the isolated execution.

One task of the processor nub 18 is to verify and load the ring-0 OS nub 16 into the isolated area, and to generate the root of a key hierarchy unique to a combination of the platform, the processor nub 18, and the operating system nub 16. The operating system nub 16 provides links to services in the primary OS 12 (e.g., the unprotected segments of the operating system), provides page management within the isolated area, and has the responsibility for loading ring-3 application modules 45, including applets 46₁ to 46_K, into protected pages allocated in the isolated area. The operating system nub 16 may also load ring-0 supporting modules.

The operating system nub 16 may choose to support paging of data between the isolated area and ordinary (e.g., non-isolated) memory. If so, then the operating system nub 16 is also responsible for encrypting and hashing the isolated area pages before evicting the page to the ordinary memory, and for checking the page contents upon restoration of the page. The isolated mode applets 46₁ to 46_K and their data are tamper-resistant and monitor-resistant from all software attacks from other applets, as well as from non-isolated-space applications (e.g., 42₁ to 42_N), dynamic link libraries (DLLs), drivers and even the primary operating system 12. Only the processor nub 18 or the operating system nub 16 can interfere with or monitor the applet's execution.

FIG. 1B is a diagram illustrating accessibility of various elements in the operating system 10 and the processor according to one embodiment of the invention. For illustration purposes, only elements of ring-0 10 and ring-3 40 are shown. The various elements in the logical operating architecture 50 access an accessible physical memory 60 according to their ring hierarchy and the execution mode.

The accessible physical memory 60 includes an isolated area 70 and a non-isolated area 80. The isolated area 70 includes applet pages 72 and nub pages 74. The non-isolated area 80 includes application pages 82 and operating system pages 84. The isolated area 70 is accessible only to elements of the operating system and processor operating in isolated execution mode. The non-isolated area 80 is accessible to all elements of the ring-0 operating system and to the processor.

The normal execution ring-0 11 including the primary OS 12, the software drivers 13, and the hardware drivers 14, can access both the OS pages 84 and the application pages 82. The normal execution ring-3, including applications 42₁ to 42_N, can access only to the application pages 82. Both the normal execution ring-0 11 and ring-3 41, however, cannot access the isolated area 70.

The isolated execution ring-0 15, including the OS nub 16 and the processor nub 18, can access to both of the isolated area 70, including the applet pages 72 and the nub pages 74, and the non-isolated area 80, including the application pages 82 and the OS pages 84. The isolated execution ring-3 45, including applets 46₁ to 46_K, can access only to the application pages 82 and the applet pages 72. The applets 46₁ to 46_K reside in the isolated area 70.

FIG. 1C is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 includes a processor 110, a host bus 120, a memory controller hub (MCH) 130, a system memory 140, an input/output controller hub (ICH) 150, a non-volatile memory, or system flash, 160, a mass storage device 170, input/output devices 175, a token bus 180, a motherboard (MB) token 182, a reader 184, and a token 186. The MCH 130 may be integrated into a chipset that integrates multiple functionalities such as the isolated execution mode, host-to-peripheral bus interface, memory control.

Similarly, the ICH 150 may also be integrated into a chipset together or separate from the MCH 130 to perform I/O functions. For clarity, not all the peripheral buses are shown. It is contemplated that the system 100 may also include peripheral buses such as Peripheral Component Interconnect (PCI), accelerated graphics port (AGP), Industry Standard Architecture (ISA) bus, and Universal Serial Bus (USB), etc.

The processor 110 represents a central processing unit of any type of architecture, such as complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), or hybrid architecture. In one embodiment, the processor 110 is compatible with an Intel Architecture (IA) processor, such as the Pentium™ series, the IA-32™ and the IA-64™. The processor 110 includes a normal execution mode 112 and an isolated execution circuit 115. The normal execution mode 112 is the mode in which the processor 110 operates in a non-secure environment, or a normal environment without the security features provided by the isolated execution mode. The isolated execution circuit 115 provides a mechanism to allow the processor 110 to operate in an isolated execution mode. The isolated execution circuit 115 provides hardware and software support for the isolated execution mode. This support includes configuration for isolated execution, definition of an isolated area, definition (e.g., decoding and execution) of isolated instructions, generation of isolated access bus cycles, and generation of isolated mode interrupts.

In one embodiment, the computer system 100 can be a single processor system, such as a desktop computer, which has only one main central processing unit, e.g. processor 110. In other embodiments, the computer system 100 can include multiple processors, e.g. processors 110, 110a, 110b, etc., as shown in FIG. 1C. Thus, the computer system 100 can be a multi-processor computer system having any number of processors. For example, the multi-processor computer system 100 can operate as part of a server or workstation environment. The basic description and operation of processor 110 will be discussed in detail below. It will be appreciated by those skilled in the art that the basic description and operation of processor 110 applies to the other processors 110a and 110b, shown in FIG. 1C, as well as any number of other processors that may be utilized in the multi-processor computer system 100 according to one embodiment of the present invention.

The processor 110 may also have multiple logical processors. A logical processor, sometimes referred to as a thread, is a functional unit within a physical processor having an architectural state and physical resources allocated according to some partitioning policy. Within the context of the present invention, the terms "thread" and "logical processor" are used to mean the same thing. A multi-threaded processor is a processor having multiple threads or multiple logical processors. A multi-processor system (e.g., the system comprising the processors 110, 110a, and 110b) may have multiple multi-threaded processors.

The host bus 120 provides interface signals to allow the processor 110 or processors 110, 100a, and 110b to communicate with other processors or devices, e.g., the MCH 130. In addition to normal mode, the host bus 120 provides an isolated access bus mode with corresponding interface signals for memory read and write cycles when the processor 110 is configured in the isolated execution mode. The isolated access bus mode is asserted on memory accesses initiated while the processor 110 is in the isolated execution

mode. The isolated access bus mode is also asserted on instruction pre-fetch and cache write-back cycles if the address is within the isolated area address range and the processor 110 is initialized in the isolated execution mode. The processor 110 responds to snoop cycles to a cached address within the isolated area address range if the isolated access bus cycle is asserted and the processor 110 is initialized into the isolated execution mode.

The MCH 130 provides control and configuration of memory and input/output devices such as the system memory 140 and the ICH 150. The MCH 130 provides interface circuits to recognize and service isolated access assertions on memory reference bus cycles, including isolated memory read and write cycles. In addition, the MCH 130 has memory range registers (e.g., base and length registers) to represent the isolated area in the system memory 140. Once configured, the MCH 130 aborts any access to the isolated area that does not have the isolated access bus mode asserted.

The system memory 140 stores system code and data. The system memory 140 is typically implemented with dynamic random access memory (DRAM) or static random access memory (SRAM). The system memory 140 includes the accessible physical memory 60 (shown in FIG. 1B). The accessible physical memory includes a loaded operating system 142, the isolated area 70 (shown in FIG. 1B), and an isolated control and status space 148. The loaded operating system 142 is the portion of the operating system that is loaded into the system memory 140. The loaded OS 142 is typically loaded from a mass storage device via some boot code in a boot storage such as a boot read only memory (ROM). The isolated area 70, as shown in FIG. 1B, is the memory area that is defined by the processor 110 when operating in the isolated execution mode. Access to the isolated area 70 is restricted and is enforced by the processor 110 and/or the MCH 130 or other chipset that integrates the isolated area functionalities. The isolated control and status space 148 is an input/output (I/O)-like, independent address space defined by the processor 110 and/or the MCH 130. The isolated control and status space 148 contains mainly the isolated execution control and status registers. The isolated control and status space 148 does not overlap any existing address space and is accessed using the isolated bus cycles. The system memory 140 may also include other programs or data which are not shown.

The ICH 150 represents a known single point in the system having the isolated execution functionality. For clarity, only one ICH 150 is shown. The system 100 may have many ICH's similar to the ICH 150. When there are multiple ICH's, a designated ICH is selected to control the isolated area configuration and status. In one embodiment, this selection is performed by an external strapping pin. As is known by one skilled in the art, other methods of selecting can be used, including using programmable configuring registers. The ICH 150 has a number of functionalities that are designed to support the isolated execution mode in addition to the traditional I/O functions. In particular, the ICH 150 includes an isolated bus cycle interface 152, the processor nub loader 52 (shown in FIG. 1A), a digest memory 154, a cryptographic key storage 155, an isolated execution logical processor manager 156, and a token bus interface 159.

The isolated bus cycle interface 152 includes circuitry to interface to the isolated bus cycle signals to recognize and service isolated bus cycles, such as the isolated read and write bus cycles. The processor nub loader 52, as shown in FIG. 1A, includes a processor nub loader code and its digest

(e.g., hash) value. The processor nub loader 52 is invoked by execution of an appropriate isolated instruction (e.g., Iso_Init) and is transferred to the isolated area 70. From the isolated area 80, the processor nub loader 52 copies the processor nub 18 from the system flash memory (e.g., the processor nub code 18 in non-volatile memory 160) into the isolated area 70, verifies and logs its integrity, and manages a symmetric key used to protect the processor nub's secrets. In one embodiment, the processor nub loader 52 is implemented in read only memory (ROM). For security purposes, the processor nub loader 52 is unchanging, tamper-resistant and non-substitutable. The digest memory 154, typically implemented in RAM, stores the digest (e.g., hash) values of the loaded processor nub 18, the operating system nub 16, and any other critical modules (e.g., ring-0 modules) loaded into the isolated execution space. The cryptographic key storage 155 holds a symmetric encryption/decryption key that is unique for the platform of the system 100. In one embodiment, the cryptographic key storage 155 includes internal fuses that are programmed at manufacturing. Alternatively, the cryptographic key storage 155 may also be created with a random number generator and a strap of a pin. The isolated execution logical processor manager 156 manages the operation of logical processors operating in isolated execution mode. In one embodiment, the isolated execution logical processor manager 156 includes a logical processor count register that tracks the number of logical processors participating in the isolated execution mode. The token bus interface 159 interfaces to the token bus 180. A combination of the processor nub loader digest, the processor nub digest, the operating system nub digest, and optionally additional digests, represents the overall isolated execution digest, referred to as isolated digest. The isolated digest is a fingerprint identifying the ring-0 code controlling the isolated execution configuration and operation. The isolated digest is used to attest or prove the state of the current isolated execution.

The non-volatile memory 160 stores non-volatile information. Typically, the nonvolatile memory 160 is implemented in flash memory. The non-volatile memory 160 includes the processor nub 18. The processor nub 18 provides the initial set-up and low-level management of the isolated area 70 (in the system memory 140), including verification, loading, and logging of the operating system nub 16, and the management of the symmetric key used to protect the operating system nub's secrets. The processor nub 18 may also provide application programming interface (API) abstractions to low-level security services provided by other hardware. The processor nub 18 may also be distributed by the original equipment manufacturer (OEM) or operating system vendor (OSV) via a boot disk.

The mass storage device 170 stores archive information such as code (e.g., processor nub 18), programs, files, data, applications (e.g., applications 42₁ to 42_N), applets (e.g., applets 46₁ to 46_K) and operating systems. The mass storage device 170 may include compact disk (CD) ROM 172, floppy diskettes 174, and hard drive 176, and any other magnetic or optical storage devices. The mass storage device 170 provides a mechanism to read machine-readable media. When implemented in software, the elements of the present invention are the code segments to perform the necessary tasks. The program or code segments can be stored in a processor readable medium or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable medium" may include any medium that can store or transfer information. Examples of the processor

readable medium include an electronic circuit, a semiconductor memory device, a ROM, a flash memory, an erasable programmable ROM (EPROM), a floppy diskette, a compact disk CD-ROM, an optical disk, a hard disk, a fiber optical medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, an Intranet, etc.

I/O devices 175 may include any I/O devices to perform I/O functions. Examples of I/O devices 175 include a controller for input devices (e.g., keyboard, mouse, trackball, pointing device), media card (e.g., audio, video, graphics), a network card, and any other peripheral controllers.

The token bus 180 provides an interface between the ICH 150 and various tokens in the system. A token is a device that performs dedicated input/output functions with security functionalities. A token has characteristics similar to a smart card, including at least one reserved-purpose public/private key pair and the ability to sign data with the private key. Examples of tokens connected to the token bus 180 include a motherboard token 182, a token reader 184, and other portable tokens 186 (e.g., smart card). The token bus interface 159 in the ICH 150 connects through the token bus 180 to the ICH 150 and ensures that when commanded to prove the state of the isolated execution, the corresponding token (e.g., the motherboard token 182, the token 186) signs only valid isolated digest information. For purposes of security, the token should be connected to the digest memory.

ISOLATED INSTRUCTIONS

FIG. 2 is a diagram illustrating a part of the isolated execution circuit 115 shown in FIG. 1C according to one embodiment of the invention. The isolated execution circuit 115 includes an execution unit 230 and a parameter storage 240.

The execution unit 230 executes an isolated instruction in the processor 110, shown in FIG. 1C. As discussed above, the processor 110 operates in a platform including the MCH 130, the system memory 140, the ICH 150, and the token bus 180 as shown in FIG. 1C. In addition, the processor 110 is configured in one of a normal execution mode and an isolated execution mode. The execution unit 230 may include a decoder to decode the isolated instruction, sub execution units to carry out the tasks in the decoded instruction, an arithmetic and logic unit (ALU) to perform arithmetic and logic functions, and associated registers. The execution unit 230 receives an isolated instruction 210.

The isolated instruction 210 may also be a macro instruction, which includes a number of micro-instructions, or assembly language level instructions. The isolated instruction 210 may also be a high level language instruction or a function, a routine, a subprogram in a high level language. In what follows, the term "instruction" refers to an abstract conception of a task or a function that is executed by the processor 110 (FIG. 1C). This task or function can be accomplished by a single operation, a series of micro-operations, a microcode routine, a macro instruction, an assembly language routine or function, a high level language statement, a high level language subroutine or subprogram or function, or a dedicated hardware circuit.

The isolated instruction 210 includes a number of instructions. In one embodiment, these instructions include an isolated initialize (iso_init) instruction 212, an isolated

close (iso_close) instruction 214, an isolated enter (iso_enter) instruction 216, an isolated exit (iso_exit) instruction 218, an isolated configuration read (iso_config_read) 222, and an isolated configuration write (iso_config_write) instruction 224. The iso_init instruction 212 may also be referred to as an isolated create (iso_create) instruction.

The parameter storage 240 contains at least one parameter to support execution of the isolated instruction 210 when the processor 110 is configured in the isolated execution mode.

The parameter storage 240 includes an internal storage 232 and an external storage 234. The internal storage 232 is internal to the processor 110 and the external storage 234 is external to the processor 110. The parameter is one of an isolated feature word, an execution mode word, a logical processor value, an isolated setting including a mask value and a base value, a frame, an exit physical address, an entry physical address, and a processor nub loader physical address.

FIG. 3 is a diagram illustrating the internal storage 232 shown in FIG. 2 according to one embodiment of the invention. The internal storage 232 includes a feature register 310, a processor control register 320, a logical processor register 330, an isolated setting register 340, a frame register set 350, an exit frame register 360, an entry frame register 370, and a loader register 380.

The feature register 310 stores the isolated feature word. The isolated feature word indicates that the isolated instructions are present in the processor 110 and which features of the isolated instructions are present. Examples of the features may include the type of isolated instructions, a random number, and a random number status. The processor control register 320 stores an execution mode word 322. The execution mode word 322 stores a bit to indicate if the processor is configured in the isolated execution mode. In one embodiment, the execution mode word is accessed only by the instructions that transfer control to or from the isolated execution mode. Examples of these instructions include the iso_init, iso_enter, and iso_exit instructions.

The logical processor register 330 stores a logical processor value 332. The logical processor value 332 indicates the number of logical processors that enter the isolated execution mode. A physical processor may have a number of logical processors, each logical processor corresponds to a logical processor. Initially, the logical processor value 332 is zero. When a logical processor enters the isolated execution mode, the logical processor value 332 increments by one. When a logical processor exits the isolated execution mode, the logical processor value decrements by one. The logical processor value 332 has a maximum value to limit the number of logical processors in the isolated execution mode. When a logical processor exits and the logical processor value 332 decrements to zero, that logical processor knows that it is the last logical processor to withdraw from the isolated execution mode.

The frame register set 350 stores a number of values corresponding to a frame. A frame includes a number of register values that need to be saved or restored when there is an exit out of or entry into an isolated execution mode.

Typically these frame values are those values that are necessary to keep track of the state of the processor. In one embodiment, the processor is a Pentium-type. The frame information includes the values of the control register 3 (CR3), control register 4 (CR4), the Interrupt Descriptor Task Register (IDTR) base, IDTR limit, the Global Descriptor Task Register (GDTR) base value, the GDTR limit value, the task register (TR) value, the extended instruction pointer

(EIP), the code segment (CS) value, the CS descriptor, the extended flags (EFLAGS), the extended stack pointer (ESP), the stack segment (SS) value, and the SS descriptor. A frame corresponding to an exit out of or entry into an isolated execution mode is referred to an exit frame or an entry frame, respectively.

The exit frame register 360 stores an exit address 362. The exit address 362 is the physical address of the exit frame. The entry frame register 370 stores an entry address 372. The entry address 372 is the physical address of the entry frame. The load register 380 stores a processor nub loader address 382. The processor nub loader address 382 is the physical address of the processor nub loader 52.

FIG. 4 is a diagram illustrating the external storage 234 shown in FIG. 2 according to one embodiment of the invention. The external storage 234 includes the MCH storage 410 and ICH storage 450.

The MCH storage 410 is in the MCH and is a part of the Isolated Area Control 135 shown in FIG. 1C. The MCH storage 410 includes an MCH logical processor register 420, an MCH mask register 430, and an MCH base register 440. The MCH logical processor register 420 stored an MCH logical processor value 422. The MCH logical processor value 422 indicates the number of logical processors that are configured to use the isolated area 70 in the system memory 140 as shown in FIG. 1C. The MCH mask register 430 and base register 440 store the MCH mask value 432 and MCH base value 442, respectively. The MCH mask value 432 and the MCH base value 442 are used to define the isolated area 70.

The ICH storage 450 is in the ICH shown in FIG. 1C. The ICH storage 450 includes an ICH logical processor register 460, an ICH mask register 470, and an ICH base register 480. The ICH logical processor register 460 stores an ICH logical processor value 462. The ICH logical processor value 462 indicates the number of logical processors that are configured to use the isolated area 70 in the system memory 140 as shown in FIG. 1C. The ICH mask register 470 and base register 480 store an ICH mask value 472 and ICH base value 482, respectively. The ICH mask value 472 and the ICH base value 482 are used to define the isolated area 70. The ICH logical processor value 462 refers to all logical processors in the register, including those in separate different physical processors. The MCH logical processor value 422, however, refers only to the logical processors in the physical processor that the corresponding MCH is directly coupled to.

FIG. 5 is a flowchart illustrating a process 500 to execute the iso_init instruction according to one embodiment of the invention.

Upon START, the process 500 prepares for the initialization (Block 510). This preparation may include verification of the protected non-paging mode, verification of the validity of the isolated area mask and base values. Then, the process 500 determines if the isolated area has been created (Block 520). If yes, the process 500 goes to Block 540. Otherwise, the process 500 creates an isolated area (Block 530). The process of creating will be described in FIG. 6A.

The process 500 then sets the range in the MCH registers and the processor registers (Block 540). This includes updating the processor mask and base registers and the MCH mask and base registers. Next, the process 500 copies the processor nub loader from the ICH into the isolated area using the processor nub loader address in the loader register (Block 550).

Then the process 500 initializes the isolated area for the platform (Block 560). Then the process 500 obtains a

location, or a physical address, of the processor nub (Block 570). Next, the process 500 transfers control to the processor nub loader by executing the processor nub loader code (Block 580). The process 500 is then terminated.

FIG. 6A is a flowchart illustrating the process 530 to execute the creating function in the iso_init instruction according to one embodiment of the invention.

Upon START, the process 530 determines if the size of the processor nub loader is less than or equal to the range of the isolated area as defined by the mask value (Block 610). If not, then the isolated area is not large enough to store the entire processor nub loader, the process 530 generates a failure or fault condition (Block 615). Otherwise, the process 530 configures the processor in the isolated execution mode by setting the appropriate bit(s) in the execution mode word (Block 620).

Then, the process 530 determines if the processor is the first processor to create the isolated area (Block 625). If not, the process 530 determines if the processor nub has been initialized (Block 630). If the processor nub has not been initialized, the process 530 generates a failure or fault condition (Block 635). If the processor nub has been properly initialized, the process 530 reads the global processor settings including the mask and base values (Block 640). Then the process 530 initialize the processor settings by transferring the global settings into the processor settings (Block 645). The process 530 is then terminated or returned to the main process.

If the processor is the first processor to create the isolated area, the process 530 determines if the mode is waiting for initialization (Block 650). This can be achieved by reading a status value from the ICH. If the status indicates that the mode is not waiting for initialization, the process 530 generates a failure or fault condition (Block 655). Otherwise, the process 530 determines if there is already an isolated area (Block 660). If there is already an isolated area, the process 530 goes to block 670. If there is not an isolated area, the process 530 saves the settings (e.g., base and mask values) in the ICH (Block 665). Then the process 530 obtains the settings (e.g., base and mask values) from the ICH (Block 670). The process 530 is then terminated or returned to the main process.

FIG. 6B is a flowchart illustrating the process 580 to execute the processor nub loader function in the iso_init instruction according to one embodiment of the invention.

Upon START, the process 580 copies the processor nub into the isolated area using the processor nub physical address in the processor nub loader (Block 675). Then the process 580 verifies the signature of the processor nub using a public key in the processor nub loader (Block 680). Next, the process 580 determines if the signature is verified (Block 685). If it is not verified, the process 580 generates a failure or fault condition (Block 690). Otherwise, the process 580 executes the processor nub (Block 695). The process 580 is then terminated or returned to the main process.

FIG. 7 is a flowchart illustrating a process 700 to execute the iso_close instruction according to one embodiment of the invention.

Upon START, the process 700 prepares for close (Block 710). The preparation may include obtaining the exit frame and saving the current IDTR limit. Then, the process 700 sets the isolated execution mode by configuring the isolated execution mode word (Block 720). Next, the process 700 removes the processor's logical processor by clearing the pointer to the entry frame of the processor (Block 730).

Then, the process 700 determines if this is the last logical processor to withdraw from the isolated execution mode

(Block 740). If not, the process 700 goes to block 760. Otherwise, the process 700 resets the isolated setting values (e.g., base and mask values) of the processor (Block 750). Then, the process 700 determines if this is the last processor to withdraw from the isolated execution mode (Block 760). If not, the process 700 goes to Block 780. Otherwise, the process 700 resets the isolated settings in the MCH and ICH (Block 770). Then, the process 700 clears the isolated execution mode by writing to the execution mode word (Block 780). The process 700 is then terminated.

FIG. 8A is a flowchart illustrating a process 800 to execute the iso_enter instruction according to one embodiment of the invention.

Upon START, the process 800 prepares for entry (Block 810). The preparation may include verification of the frame address and saving the current control register and the IDTR limit. Then, the process 800 sets the isolated execution mode by configuring the execution mode word (Block 820). Next, the process 800 stores contents of the frame register set into an exit frame (Block 830). The exit frame is pointed to by the exit address in the exit frame register. Then, the process 800 loads an entry frame into the frame register set (Block 840). The entry frame is pointed to by the entry address in the entry frame register. Then, the process 800 is terminated.

FIG. 8B is a flowchart illustrating a process 850 to execute the iso_exit instruction according to one embodiment of the invention.

Upon START, the process 850 prepares for exit (Block 860). The preparation may include verifying the frame address and clearing the IDTR limit to ensure proper dispatch to the correct handler if there is a machine check before restoring the frame. Then, the process 850 loads an exit frame into the frame register set (Block 870). The exit frame is pointed to by the exit address in the exit frame register. Then, the process 850 is terminated.

FIG. 9A is a flowchart illustrating a process 900 to execute the iso_config_read instruction according to one embodiment of the invention.

Upon START, the process 900 prepares for reading the configuration storage (Block 910). The preparation may include verifying address of the configuration storage. Then, the process 900 returns the contents of the configuration storage corresponding to the parameter storage (Block 920). Then, the process 900 is terminated.

FIG. 9B is a flowchart illustrating a process 950 to execute the iso_config_write instruction according to one embodiment of the invention.

Upon START, the process 950 prepares for writing to the configuration storage (Block 960). The preparation may include verifying address of the configuration storage. Then, the process 950 writes the contents of the configuration storage corresponding to the parameter storage (Block 970). Then, the process 950 is terminated.

While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.

What is claimed is:

1. An apparatus comprising:
an execution unit to execute an isolated instruction in a processor operating in a platform, the processor being configured in one of a normal execution mode and an isolated execution mode; and

a parameter storage containing at least one parameter to support execution of the isolated instruction when the processor is configured in the isolated execution mode; wherein the isolated instruction is one of an isolated initialize (iso_init) instruction, an isolated close (iso_close) instruction, an isolated enter (iso_enter) instruction, an isolated exit (iso_exit) instruction, an isolated configuration read (iso_config_read), and an isolated configuration write (iso_config_write) instruction.

2. The apparatus of claim 1 wherein the at least one parameter is one of an isolated feature word, an execution mode word, a logical processor value, an isolated setting including a mask value and a base value, a frame, an exit physical address, an entry physical address, and a processor nub loader physical address.

3. The apparatus of claim 2 wherein the parameter storage is one of an internal storage and an external storage.

4. The apparatus of claim 3 wherein the internal storage includes a feature register to store the isolated feature word, a control register to store the execution mode word, a logical processor register to store a logical processor value, a mask register to store the mask value, a base register to store the base value, a frame register set to store the frame, an exit frame register to store the exit address, an entry frame register to store the entry address, and a loader register to store the processor nub loader address.

5. The apparatus of claim 4 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

6. The apparatus of claim 5 wherein the execution mode word configures the processor in the isolated execution mode.

7. The apparatus of claim 6 wherein the iso_init instruction, when executed, causes the processor to:

create an isolated area in a memory if the isolated area has not been created, the isolated area being defined by the base value and the mask value;
set a range of the isolated area in the MCH storage;
copy a processor nub loader into the isolated area using the processor nub loader address in the loader register;
and
execute the processor nub loader.

8. The apparatus of claim 7 wherein the iso_init instruction, when executed, further causes the processor to:
initialize the isolated area for the platform; and
obtain a location of a processor nub.

9. The apparatus of claim 8 wherein the iso_init instruction causing the processor to execute the processor nub loader causes the processor to:
copy the processor nub into the isolated area; and
verify a signature of the processor nub using a public key contained in the processor nub loader; and

execute the processor nub if the signature is verified.

10. The apparatus of claim 6 wherein the iso_close instruction, when executed, causes the processor to:
reset the isolated setting in the processor if a corresponding logical processor is last to withdraw; and
reset the isolated setting in the MCH and the ICH if the processor is last to withdraw.

11. The apparatus of claim 6 wherein the iso_enter instruction, when executed, causes the processor to:
store contents of the frame register set in an exit frame, the exit frame being pointed to by the exit address in the exit frame register; and

load an entry frame into the frame register set, the entry frame being pointed to by the entry address in the entry frame register.

12. The apparatus of claim 6 wherein the iso_exit instruction, when executed, causes the processor to:

load an exit frame into the frame register set, the exit frame being pointed to by the exit address in the exit frame register.

13. The apparatus of claim 6 wherein the iso_config_read instruction, when executed, causes the processor to:

return contents of a configuration storage corresponding to the parameter storage.

14. The apparatus of claim 6 wherein the iso_config_write instruction, when executed, causes the processor to:

write contents of a configuration storage corresponding to the parameter storage.

15. A method comprising:

executing an isolated instruction by an execution unit in a processor operating in a platform, the processor being configured in one of a normal execution mode and an isolated execution mode; and

supporting execution of the isolated instruction by a parameter storage containing at least one parameter when the processor is configured in the isolated execution mode;

wherein executing the isolated instruction comprises executing one of an isolated initialize (iso_init) instruction, an isolated close (iso_close) instruction, an isolated enter (iso_enter) instruction, an isolated exit (iso_exit) instruction, an isolated configuration read (iso_config_read), and an isolated configuration write (iso_config_write) instruction.

16. The method of claim 15 wherein the at least one parameter is one of an isolated feature word, an execution mode word, a logical processor value, an isolated setting including a mask value and a base value, a frame, an exit physical address, an entry physical address, and a processor nub loader physical address.

17. The method of claim 16 wherein the parameter storage is one of an internal storage and an external storage.

18. The method of claim 17 wherein the internal storage includes a feature register to store the isolated feature word, a control register to store the execution mode word, a logical processor register to store a logical processor value, a mask register to store the mask value, a base register to store the base value, a frame register set to store the frame, an exit frame register to store the exit address, an entry frame register to store the entry address, and a loader register to store the processor nub loader address.

19. The method of claim 18 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

20. The method of claim 19 wherein the execution mode word configures the processor in the isolated execution mode.

21. The method of claim 20 wherein executing the iso_init instruction comprises:

creating an isolated area in a memory if the isolated area has not been created, the isolated area being defined by the base value and the mask value;

setting a range of the isolated area in the MCH storage;

copying a processor nub loader into the isolated area using the processor nub loader address in the loader register; and

executing the processor nub loader.

22. The method of claim 21 wherein executing the iso_init instruction further comprises:

initializing the isolated area for the platform; and obtaining a location of a processor nub.

23. The method of claim 22 wherein executing the processor nub loader comprises:

copying the processor nub into the isolated area; verifying a signature of the processor nub using a public key contained in the processor nub loader; and executing the processor nub if the signature is verified.

24. The method of claim 20 wherein executing the iso_close instruction comprises:

resetting the isolated setting in the processor if a corresponding logical processor is last to withdraw; and resetting the isolated setting in the MCH and the ICH if the processor is last to withdraw.

25. The method of claim 20 wherein executing the iso_enter instruction comprises:

storing contents of the frame register set in an exit frame, the exit frame being pointed to by the exit address in the exit frame register; and

loading an entry frame into the frame register set, the entry frame being pointed to by the entry address in the entry frame register.

26. The method of claim 20 wherein executing the iso_exit instruction comprises:

loading an exit frame into the frame register set, the exit frame being pointed to by the exit address in the exit frame register.

27. The method of claim 20 wherein executing the iso_config_read instruction comprises:

returning contents of a configuration storage corresponding to the parameter storage.

28. The method of claim 20 wherein executing the iso_config_write instruction comprises:

writing contents of a configuration storage corresponding to the parameter storage.

29. A system comprising:

a chipset;

a memory coupled to the chipset having an isolated memory area; and

a processor coupled to the chipset and the memory operating in a platform, the processor having an isolated instruction execution circuit, the processor being configured in one of a normal execution mode and an isolated execution mode, the isolated instruction execution circuit comprising:

an execution unit to execute an isolated instruction, and a parameter storage containing at least one parameter to support execution of the isolated instruction when the processor is configured in the isolated execution mode; wherein the isolated instruction is one of an isolated initialize (iso_init) instruction, an isolated close (iso_close) instruction, an isolated enter (iso_enter) instruction, an isolated exit (iso_exit) instruction, an isolated configuration read (iso_config_read), and an isolated configuration write (iso_config_write) instruction.

30. The system of claim 29 wherein the at least one parameter is one of an isolated feature word, an execution mode word, a logical processor value, an isolated setting including a mask value and a base value, a frame, an exit physical address, an entry physical address, and a processor nub loader physical address.

31. The system of claim 30 wherein the parameter storage is one of an internal storage and an external storage.

32. The system of claim 31 wherein the internal storage includes a feature register to store the isolated feature word, a control register to store the execution mode word, a logical processor register to store the logical processor value, a mask register to store the mask value, a base register to store the base value, a frame register set to store the frame, an exit frame register to store the exit address, an entry frame register to store the entry address, and a loader register to store the processor nub loader address. 5

33. The system of claim 32 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH. 15

34. The system of claim 33 wherein the execution mode word configures the processor in the isolated execution mode.

35. The system of claim 34 wherein the iso_init instruction, when executed, causes the processor to:

create an isolated area in a memory if the isolated area has not been created, the isolated area being defined by the base value and the mask value;

set a range of the isolated area in the MCH storage;

copy a processor nub loader into the isolated area using the processor nub loader address in the loader register; and

execute the processor nub loader.

36. The system of claim 35 wherein the iso_init instruction, when executed, further causes the processor to:

initialize the isolated area for the platform; and

obtain a location of a processor nub.

37. The system of claim 36 wherein the iso_init instruction causing the processor to execute the processor nub loader causes the processor to:

copy the processor nub into the isolated area; and

verify a signature of the processor nub using a public key contained in the processor nub loader; and

execute the processor nub if the signature is verified.

38. The system of claim 34 wherein the iso_close instruction, when executed, causes the processor to:

reset the isolated setting in the processor if a corresponding logical processor is last to withdraw; and

reset the isolated setting in the MCH and the ICH if the processor is last to withdraw.

39. The system of claim 34 wherein the iso_enter instruction, when executed, causes the processor to:

store contents of the frame register set in an exit frame, the exit frame being pointed to by the exit address in the exit frame register; and

load an entry frame into the frame register set, the entry frame being pointed to by the entry address in the entry frame register.

40. The system of claim 34 wherein the iso_exit instruction, when executed, causes the processor to:

load an exit frame into the frame register set, the exit frame being pointed to by the exit address in the exit frame register.

41. The system of claim 34 wherein the iso_config_read instruction, when executed, causes the processor to:

return contents of a configuration storage corresponding to the parameter storage.

42. The system of claim 34 wherein the iso_config_write instruction, when executed, causes the processor to:

write contents of a configuration storage corresponding to the parameter storage.

43. A computer program product comprising:
a machine readable medium having computer program code embodied therein, the computer program product having:

computer readable program code for executing an isolated instruction by an execution unit in a processor operating in a platform, the processor being configured in one of a normal execution mode and an isolated execution mode; and

computer readable program code for supporting execution of the isolated instruction by a parameter storage containing at least one parameter when the processor is configured in the isolated execution mode; wherein the computer readable program code for executing the isolated instruction comprises computer readable program code for executing one of an isolated initialize (iso_init) instruction, an isolated close (iso_close) instruction, an isolated enter (iso_enter) instruction, an isolated exit (iso_exit) instruction, an isolated configuration read (iso_config_read), and an isolated configuration write (iso_config_write) instruction.

44. The computer program product of claim 43 wherein the at least one parameter is one of an isolated feature word, an execution mode word, a logical processor value, an isolated setting including a mask value and a base value, a frame, an exit physical address, an entry physical address, and a processor nub loader physical address.

45. The computer program product of claim 44 wherein the parameter storage is one of an internal storage and an external storage.

46. The computer program product of claim 45 wherein the internal storage includes a feature register to store the isolated feature word, a control register to store the execution mode word, a logical processor register to store a logical processor value, a mask register to store the mask value, a base register to store the base value, a frame register set to store the frame, an exit frame register to store the exit address, an entry frame register to store the entry address, and a loader register to store the processor nub loader address.

47. The computer program product of claim 46 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

48. The computer program product of claim 47 wherein the execution mode word configures the processor in the isolated execution mode.

49. The computer program product of claim 48 wherein the computer readable program code for executing the iso_init instruction comprises:

computer readable program code for creating an isolated area in a memory if the isolated area has not been created, the isolated area being defined by the base value and the mask value;

computer readable program code for setting a range of the isolated area in the MCH storage;

computer readable program code for copying a processor nub loader into the isolated area using the processor nub loader address in the loader register; and

computer readable program code for executing the processor nub loader.

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,507,904 B1
DATED : January 14, 2003
INVENTOR(S) : Ellison et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 1,
Line 16, delete "modem" and insert -- modern --.

Signed and Sealed this

Eighth Day of April, 2003



JAMES E. ROGAN
Director of the United States Patent and Trademark Office